Р. Данкан

Профессиональная работа в MS-DOS

Издательство «Мир»

Есть такая фирма!

IBM-совместимые персональные компьютеры по самым стабильным и привлекательным в России ценам.

Широкий выбор моделей на базе процессоров **286, 386DX, 486DX** и периферийных устройств.

Только **надежные комплектующие** из США, Германии, Японии, Сингапура и Тайваня плюс высокое качество сборки.

Гарантийное обслуживание - залог уверенности в "завтрашнем дне" Вашей техники.

Доставка купленной Вами техники до любого московского аэропорта.

НАУЧНО-ТЕХНИЧЕСКОЕ АКЦИОНЕРНОЕ ОБЩЕСТВО "ЛЭНД"

г.Москва, метро "Домодедовская"
ул.Воронежская, д.24, строение 2,
АО "ЛЭНД"

телефоны: 398-49-54, 398-49-66, 398-44-81
телефакс: 398-49-66

MegaPro

Профессиональная работа в MS-DOS

Ray Duncan

Advanced MSDOS Programming

The Microsoft® guide for Assembly Language and C programmers

Microsoft PRESS®

Р. Данкан

Профессиональная работа в MS-DOS

Перевод с английского
канд. физ.-мат. наук Ю. И. Малахова,
канд. пед. наук К. Г. Финогенова

под редакцией
канд. пед. наук К. Г. Финогенова

Москва «Мир» 1993

ББК 32.97
Д18
УДК 681.03.06

Данкан Р.
Д18 Профессиональная работа в MS-DOS: Пер. с англ.– М.: Мир, 1993.– 509 с., ил.
ISBN 5-03-002142-6

Книга является переводом второго издания всемирно известной монографии, выпущенной фирмой «Майкрософт». Описаны базовая система ввода-вывода (BIOS) для IBM PC, XT/AT, PS/2 и их периферии, операционная система MS-DOS до версии 4, инструментарий MS-DOS, вопросы совместимости с OS/2 и многое другое. Книга содержит большое количество листингов на языке ассемблера и Си, представляющих собой как фрагменты программ, так и полные утилиты.

Для профессиональных программистов, работающих с языком ассемблера и Си.

Д $\frac{\textbf{2402090000-012}}{\textbf{041 (01)-93}}$ **123-92** **ББК 32.97**

Редакция литературы по информатике

ISBN 5-03-002142-6 (русск.)
ISBN 1-55615-157-8 (англ.)

© 1986, 1988 by Ray Duncan
© перевод на русский язык, Малахов Ю. И., Финогенов К. Г., 1993

ПРЕДИСЛОВИЕ ПЕРЕВОДЧИКА

Предлагаемая читателю книга рассчитана на тех пользователей персональных компьютеров типа IBM PC, которые хотели бы на профессиональном уровне разобраться в возможностях операционной системы (ОС) MS-DOS и научиться использовать системные средства в своих программах.

Описанные в книге системные сервисные программы, или функции ОС, служащие для выполнения элементарных действий по управлению оборудованием компьютера и организации вычислительного процесса, относятся к числу важнейших компонентов ОС. С их помощью осуществляется обслуживание файлового хозяйства: создание, переименование, поиск и удаление файлов, запись и чтение данных, операции с дисками и каталогами, обращение к системным областям ОС, получение информации о дисковой системе и многое другое. Большая группа функций ОС предназначена для ввода информации с клавиатуры, обеспечивая такие возможности, как ввод символов с эхо и без него, анализ расширенных кодов ASCII и скен-кодов клавиш, ввод с предварительной очисткой входного буфера, синхронный и асинхронный ввод, настройка клавиатуры. Сервисные функции обслуживания видеотерминала позволяют выводить на экран цветные символы и символьные строки, организовывать дисплейные окна с прокруткой изображения, переключать видеостраницы, настраивать цветовую палитру. С помощью системных функций осуществляются динамическое выделение и освобождение памяти, управление расширенной и дополнительной памятью, а также динамический запуск дочерних программ, что позволяет организовывать иерархические программные комплексы. В таких комплексах родительский процесс в зависимости от конкретных условий инициирует те или иные дочерние процессы, те в свою очередь вызывают к жизни процессы следующего уровня и т. д. Сюда же примыкают системные функции для создания программ, резидентных в памяти,– важнейшего элемента идеологии программного обеспечения персональных компьютеров. Наконец, системные средства используются при разработке драйверов и обработчиков прерываний.

Для программистов, работающих на языке ассемблера, функции ОС являются основным инструментальным средством, без которого практически не обходится ни одна программа. При разработке программ на языках высокого уровня (Паскаль, Си) необходимость в прямом использовании системных функций возникает реже – лишь если средства языка не обеспечивают решения поставленных задач. Однако в любом случае знакомство с функциями ОС является необходимым элементом подготовки квалифицированного программиста, поскольку, как видно из приведенного выше перечня, состав функций ОС в значительной степени определяет ориентацию и возможности ОС и ее пригодность для решения конкретных прикладных задач.

Книга написана на хорошем профессиональном уровне и в то же время отличается простотой изложения, ясностью языка и продуманным расположением материала.

Конечно, она рассчитана не на начинающих. От читателя требуется знакомство с вычислительной техникой вообще и с машинами типа IBM PC в частности, а также умение разбираться в программах на языках ассемблера и Си. Впрочем, практически все понятия, вводимые в книге, объясняются достаточно подробно.

Книга с равным успехом может служить и учебным пособием при первом ознакомлении с предметом, и справочником при практическом программировании (заметим, что обилие функций ОС – их общее число превышает 400 – не позволяет работать с ними без настольного справочника). Раздел 1, занимающий по объему больше половины книги, содержит систематическое описание основных возможностей MS-DOS, иллюстрированное большим количеством несложных программ на языках ассемблера и Си. В разд. 2, 3 и 4 приведены исчерпывающие справочные данные (часто также с программными примерами) по функциям ОС, включая функции базовой системы ввода-вывода BIOS, манипулятора «мышь» и спецификации дополнительной памяти Lotus/Intel/Microsoft EMS.

В целом книга, несомненно, будет незаменимым практическим руководством для самых широких кругов программистов.

К. Г. Финогенов

ВВЕДЕНИЕ

Книга «Профессиональная работа в MS-DOS» предназначается программистам, работающим на языках Си и ассемблера. Она содержит всю информацию, необходимую для написания помехоустойчивых, высокопроизводительных прикладных программ в среде операционной системы MS-DOS. Так как я полагаю, что работающие, хорошо документированные программы – это лучшие во всех отношениях обучающие средства, то включил в книгу примеры, содержащие детальные тексты программ, в том числе и утилит, которые читатель может адаптировать под свои задачи.

Книга, которая одновременно является и учебником, и справочником, разделена на четыре раздела, что поможет читателю находить интересующую его информацию. В разд. 1 обсуждаются представленные по функциональным группам возможности MS-DOS для программирования таких общих действий пользователя, как ввод информации, управление дисплеем, управление памятью, работа с файлами. Программам специального назначения, таким, как обработчики прерываний, драйверы внешних устройств и фильтры, отведены отдельные главы.

Раздел 2 – это полный справочник по функциям MS-DOS, устроенный таким образом, что вы сразу можете видеть, как вызывается функция, результаты ее выполнения и номер версии, где она впервые была применена. Кроме того, я включил замечания, касающиеся специального использования функций, а также ссылок к соответствующим функциям. Для каждой функции в разд. 2 приведены примеры программ на языке ассемблера.

Разделы 3 и 4 касаются функций IBM ПЗУ BIOS, драйвера устройства ввода «мышь» фирмы Microsoft и функций спецификации дополнительной памяти Lotus/Intel/Microsoft. Формат записей в этих двух разделах точно такой же, как и в разд. 2, за исключением того, что в некоторых примерах не приведены программы.

Представленные в этой книге программы написаны с помощью изумительного текстового редактора Brief системы Solution Systems. Затем они транслировались с помощью Макроассемблера версии 5.1 фирмы Microsoft или компилировались с помощью компилятора с языка Си версии 5.1 фирмы Microsoft. Они были испытаны в среде MS-DOS версий 2.1, 3.1, 3.3 и 4.0 на компьютерах IBM PC с процессором 8088, IBM PC/AT с процессором 80286, IBM PS/2 модели 80 с процессором 80386. Насколько мне известно, у них нет программной или аппаратной зависимости, которые препятствовали бы их выполнению на машинах, совместимых с IBM PC, в среде MS-DOS версии 2.0 или старше.

ДОПОЛНЕНИЯ КО ВТОРОМУ ИЗДАНИЮ

Те читатели, которые хорошо знакомы с первым изданием, во втором найдут много изменений, хотя структура книги осталась прежней. Удалена большая часть материала, где система MS-DOS сравнивалась с системами CP/M и UNIX/XENIX. Еще несколько лет тому назад такое сравнение было полезным, однако теперь система MS-DOS стала универсальной и заслуживает самостоятельного рассмотрения.

В прошлом единая глава разделена на три более однородные по содержанию главы, в которых рассматриваются вопросы работы клавиатуры и устройства ввода «мышь», дисплея, последовательного порта и принтера. Вопросы аппаратной зависимости средств визуализации изображения специальным образом не рассматривались; хотя эта тема теперь даже важнее, чем когда-либо раньше, однако она столь сложна и многогранна, что требует отдельной книги. Появилась новая глава, в которой обсуждаются вопросы совместимости и переносимости прикладных программ системы MS-DOS, а также дано краткое введение в новую многозадачную операционную систему Microsoft OS/2, обеспечивающую защищенный режим работы.

Приведенные в конце книги справочники значительно обновлены и дополнены материалами, касающимися системы MS-DOS вплоть до версии 4.0, ПЗУ BIOS компьютера IBM PS/2 модели 80, видеоадаптера VGA, драйвера устройства ввода «мышь» фирмы Microsoft версии 6.0, а также спецификации дополнительной памяти Lotus/Intel/Microsoft версии 4.0.

За два года, прошедшие с момента выхода в свет первого издания книги «Профессиональная работа в MS-DOS», сотни читателей любезно прислали мне свои замечания и предложения, многие из которых я постарался учесть в этом новом издании. Как и прежде, пожалуйста, связывайтесь со мной через MCI MAIL (пользовательское имя LMI), CompuServe (пользовательский идентификатор 72406, 1577) или BIX (пользовательское имя rduncan).

Рей Данкан, Лос-Анджелес,
Сентябрь 1988

Посвящается Каролин

Благодарности

Мои слова глубокой благодарности – талантливым редакторам и производственному персоналу Microsoft Press, усилиями которых была выпущена эта книга, а также блестящим разработчикам фирмы Microsoft, которые создали замечательные программы и тем самым обеспечили мне возможность написать книгу. Слова особой благодарности – Майку Хелверсону, Джеффу Хинч, Мэри Энн Джонс, Клодетте Мур, Доре Шеттак и Марку Збиковски, так как если эта книга чего-нибудь стоит, то в значительной мере благодаря этим людям.

Раздел 1. Программирование в MS-DOS

ГЛАВА 1. ПРОИСХОЖДЕНИЕ MS-DOS

Всего за семь лет система MS-DOS прошла путь от простого загрузчика до универсальной сложившейся операционной системы для персональных компьютеров, построенных на базе микропроцессоров Intel 8086 (рис. 1.1). MS-DOS поддерживает компьютерные сети и графические интерфейсы пользователя, всевозможные запоминающие устройства, служит основой для тысяч прикладных программ. Система MS-DOS, имеющая более 10 млн. зарегистрированных пользователей, постоянно «отбирает» пользователей у своих конкурентов.

Предшественником MS-DOS была операционная система 86-DOS, написанная в середине 80-х гг. Тимом Петерсоном для компании Seattle Computer Products. В то время наиболее популярной системой для микрокомпьютеров на базе Intel 8080 и Zilog Z-80 была операционная система CP/M-80 фирмы Digital Research. Эта система обеспечивала доступ к разнообразным средствам прикладного программного обеспечения (текстовые процессоры, администраторы баз данных и т. д).

Для облегчения процесса переноса прикладных программ из 8-бит системы CP/M-80 в новую 16-бит среду системы 86-DOS последняя изначально строилась так, чтобы в ней имитировались все функции и виды операций CP/M-80. Вследствие этого структуры блоков управления файлами, префиксов сегментов программ и выполнимых файлов в системе 86-DOS почти идентичны структурам CP/M-80. Программы, существовавшие в CP/M-80, можно было легко преобразовать (обрабатывая файлы исходных программ с помощью специального транслятора) и далее запускать в системе 86-DOS либо сразу, либо выполнив несложное ручное редактирование.

Ввиду того что 86-DOS поставлялась на рынок как собственная операционная система семейства микрокомпьютеров фирмы Seattle Computer Product с интерфейсом S-100 на базе Intel 8086, в целом такой подход слабо повлиял на состояние дел в мире персональных компьютеров. Другие поставщики микрокомпьютеров на базе Intel 8086, вынужденные по очевидным причинам применять операционную систему конкурентов, с нетерпением ждали выпуска системы CP/M-86 фирмы Digital Research.

В октябре 1980 г. компания IBM предложила фирмам, занимающимся разработкой программного обеспечения для микрокомпьютеров, начать поиск операционной системы для нового семейства персональных компьютеров. Фирма Microsoft не могла предложить собственной операционной системы, за исключением автономной версии Microsoft BASIC, однако она заплатила фирме Seattle Computer Products за право продавать систему Петерсона 86-DOS. За это Seattle Computer Products получила лицензию на право использовать и продавать языки программирования и все версии операционной системы для микропроцессора 8086, разработанные фирмой Microsoft. В июле 1981 г. Microsoft приобрела все права на систему 86-DOS, значительно переработала ее и дала название MS-DOS. Когда осенью 1981 г. появились первые компьютеры IBM PC, фирма IBM предложила для них в качестве основной операционную систему MS-DOS, названную PC DOS 1.0. Кроме того, фирма IBM выбрала для микрокомпьютеров PC в качестве альтернативных операционных систем системы CP/M-86 (фирмы Digital Research) и P-system (фирмы Softech). Однако обе эти системы

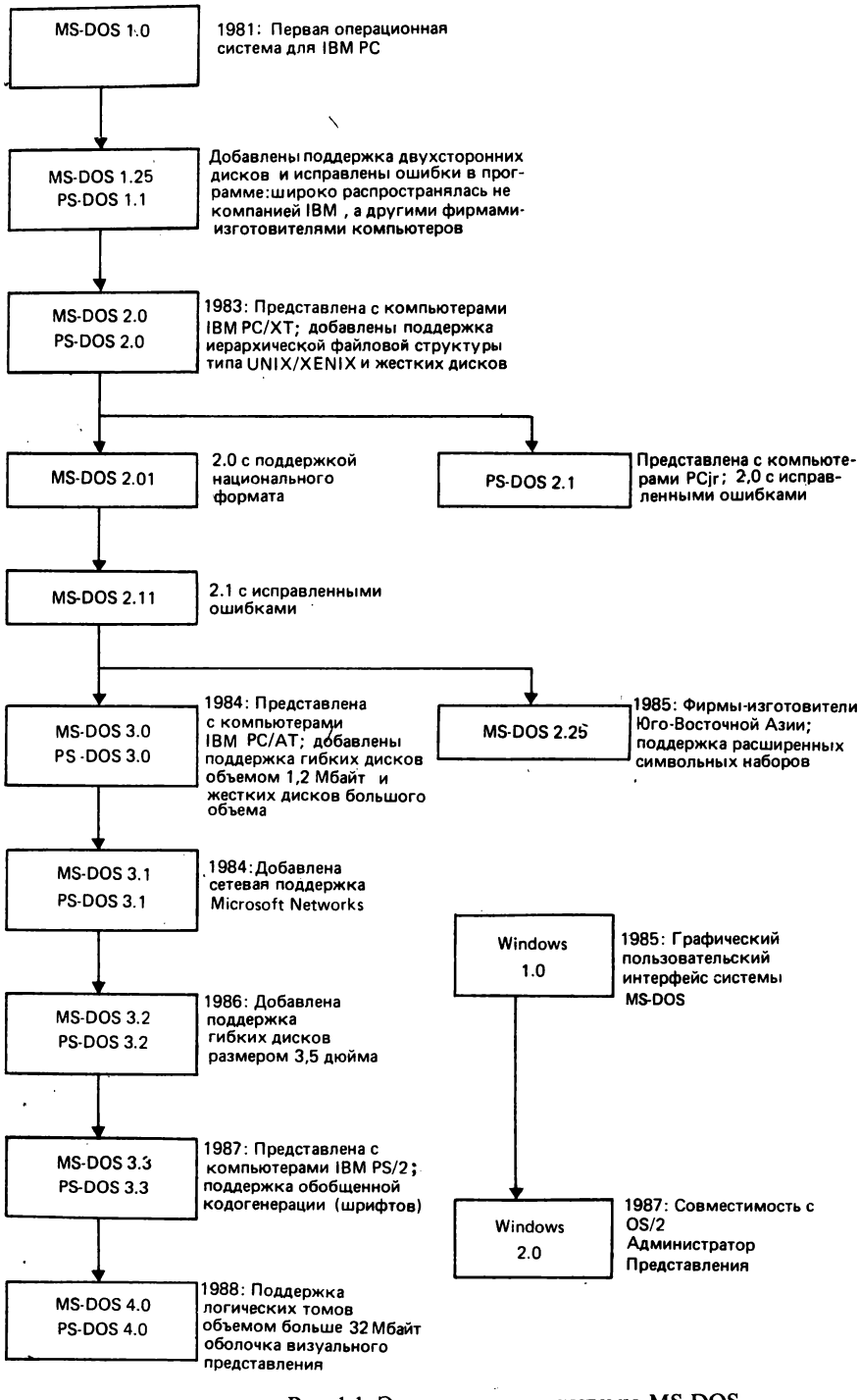


Рис. 1.1. Этапы развития системы MS-DOS.

имели ряд недостатков: обладали малым для IBM PC быстродействием, высокой стоимостью, отсутствием доступных языков программирования. Окончательно чаша весов склонилась в пользу системы PC-DOS после того, как фирма IBM с ее помощью реализовала все прикладные программные средства для IBM PC, а также инструментарий, работающий под их управлением. Поэтому с самого начала разработчики программного обеспечения ориентировались на PC-DOS, а системы CP/M-86 и P-system не заняли сколько-нибудь значительного места на рынке программного обеспечения для IBM PC.

Несмотря на сходство со своим прародителем CP/M-80, система MS-DOS (версия 1.0) содержит ряд существенных отличий, в том числе:

- Более совершенная структура каталога диска, включающая информацию об атрибутах файла (является ли, например, файл системным или скрытым), его точную длину в байтах, дату создания или последней модификации.
- Лучшее использование дискового пространства, а также метод управления, обеспечивающий исключительно быстрый последовательный или прямой доступ и загрузку программ.
- Расширенный набор сервисных программ операционной системы, включая аппаратно-независимые функции установки или чтения даты и времени, интерпретатор командной строки, операции с многоблочными записями ввода-вывода и записи переменного размера.
- Командный файл AUTOEXEC.BAT для выполнения последовательности команд пользователя при начальной или повторной загрузке системы.

Фирма IBM была единственным крупным производителем компьютеров, которая оснастила свою продукцию системой MS-DOS (версия 1.0), названной PC-DOS 1.0. Система MS-DOS версии 1.25 (эквивалентная системе фирмы IBM PC-DOS 1.1) была опубликована в июне 1982 г. В ней исправлено несколько ошибок. Кроме того, она поддерживала работу двухсторонних дисков и более полную аппаратную независимость ядра DOS. Эту версию системы DOS использовали в своей продукции кроме IBM и другие фирмы – Texas Instruments, COMPAQ, Columbia, которые вышли на рынок персональных компьютеров раньше. Из-за резкого снижения цен на ОЗУ и жесткие диски популярность системы MS-DOS (версия 1) была недолгой.

Система MS-DOS версии 2.0 (эквивалентная PC-DOS 2.0) впервые опубликована в марте 1983 г. Оглядываясь назад, можно сказать, что это была принципиально новая операционная система, хотя при ее создании было сделано все возможное для сохранения совместимости с системой MS-DOS 1.0. Эта система содержала много принципиальных новшеств и отличительных черт, в том числе:

- Поддержка гибких дисков большого объема и жестких дисков.
- Большое количество черт, напоминающих системы UNIX/XENIX, включая иерархическую структуру файлов, дескрипторы файлов, перенаправление ввода-вывода, конвейеры и фильтры.
- Фоновая печать (буферизация печати).
- Метки тома, а также дополнительные атрибуты файлов.
- Устанавливаемые драйверы внешних устройств.
- Настраиваемый пользователем файл конфигурирования системы, который осуществляет управление загрузкой дополнительных драйверов внешних устройств, числом системных буферов диска и т. д.
- Поддержка блоков окружения (контекстных блоков), которые можно использовать для обмена информацией между программами.
- Необязательный драйвер дисплея ANSI, который позволяет программам задавать положение курсора и осуществляет управление характеристиками дисплея аппаратно-независимым образом.

- Обеспечение динамического распределения памяти, ее модификации и освобождения прикладными программами.
- Поддержка настраиваемых пользователем командных интерпретаторов (оболочек).
- Системные таблица для поддержки прикладного программного обеспечения при модификации так называемых национальных форматов (обозначение денежных знаков, времени и даты).

Представленная позже система MS-DOS версии 2.11 существенно улучшила использование национальных форматов (табличное управление символами денежных знаков, форматы даты, символы десятичной точки, разделители денежных знаков и т. д.). В ней поддерживаются 16-бит представления иероглифов, исправлено несколько ошибок. Версия 2.11 быстро стала базовой для персональных компьютеров с процессором 8086/8088, выпуск которых был освоен всеми основными фирмами – изготовителями комплексного оборудования, включая Hewlett-Packard, Wang, Digital Equipment Corporation, Texas Instruments, COMPAQ и Tandy.

Система MS-DOS версии 2.25, опубликованная в сентябре 1985 г., получила широкое распространение в странах Юго-Восточной Азии, но никогда не поставлялась в США и страны Европы. В этой версии расширено национальное обеспечение в части японских и корейских символов, исправлены некоторые ошибки. Многие системные утилиты стали совместимыми с системой MS-DOS (версия 3.0).

Система MS-DOS версии 3.0 впервые представлена фирмой IBM в августе 1984 г. в момент начала выпуска компьютеров IBM PC/AT на базе процессора 80286. Это была еще одна во многом переписанная операционная система, обладавшая новыми важными особенностями:

- Непосредственное управление спулером печати прикладными программами.
- Дальнейшее расширение национальных форматов при обозначении денежных знаков.
- Расширенные сообщения об ошибках, включая код ошибки, который предлагает прикладной программе методику ее исправления.
- Поддержка захвата и разделения файлов и записей.
- Поддержка жестких дисков большого объема.

Система MS-DOS версии 3.1, опубликованная в ноябре 1984 г., оснащена средствами разделения файлов и устройств печати через локальную сеть. Начиная с версии 3.1, новый модуль операционной системы, названный модулем перенаправления, перехватывает запросы прикладных программ на ввод-вывод и фильтрует из них те, которые направлены к устройствам локальной сети, пропуская эти запросы к другой машине для выполнения.

Модификации системы MS-DOS (3.1 и старше) носят скорее эволюционный, чем революционный характер. В версии 3.2, которая появилась в 1986 г., обобщено определение драйверов устройств, что значительно упростило средства поддержки новых магнитных носителей (таких, как гибкий диск размером 3,5 дюйма). Система MS-DOS (версия 3.3), опубликованная в 1987 г. одновременно с началом выпуска персональных компьютеров IBM нового семейства – PS/2, оснащена еще более мощными средствами поддержки работы клавиатуры на нескольких иностранных языках, включая соответствующие наборы символов печатающего устройства и шрифтов дисплея. Возможности версии 4.0, впервые представленной в 1988 г., значительно возросли благодаря оболочке визуального представления и средствам поддержки файловых систем очень больших размеров.

Одновременно с развитием системы MS-DOS фирма Microsoft вела интенсивные разработки в области пользовательских интерфейсов и многозадачных операционных систем. Пакет Microsoft Windows, впервые примененный в 1985 г., позволил MS-DOS

реализовать многозадачный режим и режим графического «рабочего стола» пользователя. Пакет Windows завоевал широкую популярность среди разработчиков сложных прикладных графических программ, таких, как система подготовки издательских оригиналов, система автоматизированного проектирования (САПР). Последнее объясняется тем, что пакет позволяет использовать в программах все достоинства любого имеющегося в наличии устройства вывода без какой-либо аппаратной зависимости.

Операционная система фирмы Microsoft MS OS/2, опубликованная в 1987 г., представляет собой новый стандарт для прикладных программистов. В нем реализованы такие возможности, как защищенный и многозадачный режимы, система виртуальной памяти. Система создана специально для выполнения прикладных программ, требующих средств быстрой графики, подключения к локальной сети и организации взаимодействия процессов. Несмотря на то что система MS OS/2 является новым продуктом, а не производной от MS-DOS, ее пользовательский интерфейс и файловая система совместимы с MS-DOS и Microsoft Windows. Система позволяет выполнять прикладную программу реального режима (MS-DOS) вместе с программами защищенного режима MS OS/2. Эта совместимость дает возможность пользователям без особых трудностей переходить от операционной среды MS-DOS к среде OS/2 и обратно.

Каковы перспективы развития системы MS-DOS? Однозначно ответить на этот вопрос могут только члены групп перспективного планирования фирм Microsoft и IBM. Однако без особого риска ошибиться можно предположить, что система MS-DOS, требующая относительно небольшой памяти, легко адаптируемая к разнообразным аппаратным конфигурациям и имеющая громадное число пользователей, останется популярной среди программистов и издателей программного обеспечения в течение ближайшего будущего.

ГЛАВА 2. СИСТЕМА MS-DOS В ДЕЙСТВИИ

Маловероятно, чтобы вас когда-нибудь позвали конфигурировать систему MS-DOS для новой модели компьютера. Тем не менее знакомство с общей структурой системы MS-DOS часто очень полезно для понимания поведения вычислительной системы в целом. В этой главе мы обсудим вопросы организации MS-DOS и ее загрузки в память в момент включения компьютера.

СТРУКТУРА MS-DOS

Система MS-DOS разбита на несколько уровней, которые служат для отделения логики ядра операционной системы и восприятия системы пользователем от технических средств, реализующих ее работу. К таким уровням относятся:

- BIOS (базовая система ввода-вывода).
- Ядро системы DOS.
- Командный процессор (оболочка).

Мы обсудим отдельно функции каждого из указанных уровней.

Модуль BIOS

Модуль BIOS индивидуален для каждой вычислительной системы и поставляется ее изготовителем. В этом модуле по умолчанию резидентно содержатся аппаратно-зависимые драйверы следующих устройств:

- Консольный дисплей с клавиатурой (CON).
- Устройство построчной печати (PRN).

- Последовательный канал связи (AUX).
- Часы/календарь (CLOCK$).
- Дисковое устройство начальной загрузки (блочно-ориентированное устройство).

Ядро системы MS-DOS взаимодействует с драйверами этих устройств с помощью пакетов запросов ввода-вывода. Затем драйверы переводят эти запросы в сами команды для различных аппаратных контроллеров. Во многих системах MS-DOS, включая IBM-PC, самые простые компоненты драйверов аппаратуры размещаются в постоянном запоминающем устройстве (ПЗУ), так что их по очереди могут использовать автономные прикладные программы, средства диагностики ошибок и программа запуска системы.

Термины «резидентный» и «устанавливаемый» используются для того, чтобы различать драйверы, встроенные в BIOS, и драйверы, устновленные при инициализации системы командами DEVICE в файле CONFIG.SYS. (Устанавливаемые драйверы будут подробнее обсуждаться дальше в этой главе и гл. 14.)

Модуль BIOS считывается в основную память во время инициализации системы как часть файла с именем I0.SYS. (В системе PC-DOS этот файл именуется IBMBIO.COM.) Этот файл снабжается специальными атрибутами *скрытый* и *системный*.

Ядро DOS

Ядро DOS реализует MS-DOS, как она видится прикладным программам. Ядро – это специальная программа, поставляемая фирмой Microsoft Corporation, которая включает набор аппаратно-независимых сервисных программ, называемых системными функциями. К ним относятся следующие функции:

- Управление файлами и записями.
- Управление памятью.
- Символьно-ориентированное устройство ввода-вывода.
- Порождение (spawning) других задач.
- Доступ к часам реального времени.

Программы могут обращаться к системным функциям путем загрузки регистров параметрами функций и последующей передачи управления операционной системе с помощью программного прерывания.

Ядро DOS считывается в память при инициализации системы из файла MSDOS.SYS, находящегося на диске начальной загрузки. (Этот файл в системе PC-DOS называется IBMDOS.COM.) Этот файл отмечается атрибутами *скрытый* и *системный*.

Командный процессор

Командный процессор, или оболочка, – это интерфейс пользователя с операционной системой. Он отвечает за анализ синтаксиса и выполнение команд пользователя, включая загрузку и выполнение других программ, находящихся на диске или другом массовом запоминающем устройстве.

Стандартная оболочка, поставляемая вместе с системой MS-DOS, находится в командном файле с именем COMMAND.COM. Хотя часто COMMAND.COM воспринимается пользователем как сама система MS-DOS, важно осознавать, что COMMAND.COM не является операционной системой, а представляет собой специальный класс программ, выполняемых под управлением MS-DOS. Файл COMMAND.COM можно заменить оболочкой, созданной самим программистом добавлением ее с помощью директивы SHELL в файл конфигурирования системы (CONFIG.SYS), расположенный на том носителе, с которого система запускается. Программный продукт COMMAND-PLUS системы ESP systems – пример альтернативной оболочки.

Еще о файле COMMAND.COM

Стандартная оболочка системы MS-DOS, COMMAND.COM, делится на три части:

- Резидентная часть
- Секция инициализации
- Транзитный модуль

Резидентная часть загружается в младшие адреса памяти над ядром системы DOS, ее буферами и таблицами. Она содержит программы обработки управляющих клавиш Ctrl-C и Ctrl-Break, критических ошибок и завершения (окончательного выхода) других транзитных программ. Эта часть COMMAND.COM выдает сообщения об ошибках и запрос пользователю

Abort, Retry, Ignore? (Прервать, повторить, игнорировать?)

Резидентная часть содержит также программу, необходимую для выполнения, когда это потребуется, перезагрузки транзитной части COMMAND.COM.

Секция инициализации файла COMMAND.COM загружается над его резидентной частью в момент начала работы системы. Она обрабатывает командный файл AUTOEXEC.BAT (список команд пользователя, предназначенный для выполнения при загрузке системы), если он существует, а потом отбрасывает его.

Транзитная часть COMMAND.COM загружается по старшим адресам памяти, причем отводимое ей пространство может использоваться прикладными программами для собственных целей. Транзитный модуль выдает запрос пользователю, считывает команды с клавиатуры терминала или из командного файла и обеспечивает их выполнение. По завершению работы прикладной программы резидентная часть файла COMMAND.COM подсчитывает контрольную сумму транзитного модуля для определения возможной его порчи, а в случае необходимости загружает с диска его новую копию.

Команды пользователя, принимаемые COMMAND.COM, разделяются на три категории:

- Внутренние команды
- Внешние команды
- Командные файлы

Внутренние команды выполняются с помощью программ, находящихся в самом файле COMMAND.COM. К ним относятся такие команды, как COPY, REN(AME), DIR(ECTORY) и DEL(ETE). Программы для внутренних команд включены в транзитную часть файла COMMAND.COM.

Внешние команды, называемые также транзитными программами, представляют собой имена программ, находящихся в файлах на диске. Прежде чем выполнить эти программы, небходимо их загрузить с диска в область памяти для транзитных программ (transient program area, TPA) (см. раздел этой главы «Загрузка системы MS-DOS»). Типичными примерами внешних команд являются CHKDSK, BACKUP и RESTORE. Сразу после завершения своей работы внешняя команда удаляется из памяти, и, следовательно, при каждом ее вызове такую команду необходимо перезагружать с диска.

Командные файлы – это текстовые файлы, которые содержат списки внутренних или внешних команд, а также свои специальные внутренние команды. Эти файлы обрабатываются специальным интерпретатором, который встроен в транзитную часть COMMAND.COM. Интерпретатор построчно считывает командный файл и последовательно выполняет каждую из заданных операций.

Для выполнения команды пользователя COMMAND.COM вначале выясняет,

относится ли введенное имя к встроенной (внутренней) команде, которая выполняется непосредственно. Если нет, то ищется внешняя команда (выполнимый программный файл) или командный файл с тем же именем. Поиск осуществляется сначала в текущем каталоге текущего дисковода, а затем в каталогах, указанных в последней команде PATH. В каждом просматриваемом каталоге файл COMMAND.COM ищет сначала файл с расширением .COM, затем .EXE и потом .BAT. Если при поиске не обнаружено файла ни одного из трех указанных типов, то COMMAND.COM выдает на экран дисплея сообщение

Bad command or file name (Неверно задана команда или имя файла)

Если найден файл с расширением .COM или .EXE, то COMMAND.COM, используя системную функцию EXEC, осуществляет загрузку и выполнение найденного файла. Функция EXEC строит специальную структуру данных, которая называется префиксом сегмента программы (program segment prefix, PSP), над резидентной частью COMMAND.COM в транзитной программной области. Префикс содержит различные связи и указатели, необходимые прикладной программе. Далее функция EXEC загружает саму программу непосредственно над префиксом и выполняет небходимую настройку адресов. В завершение функция EXEC устанавливает соответствующие регистры и передает управление на точку входа программы. (Детальнее префикс PSP и функция EXEC обсуждаются в гл. 3 и 12.) После завершения выполнения транзитной программы вызывается специальная системная функция завершения, которая в свою очередь освобождает память от транзитной программы и возвращает управление программе, вызвавшей загрузку транзитной программы (в данном случае COMMAND.COM).

Транзитная программа в процессе своего выполнения имеет почти полный контроль над ресурсами системы. Кроме нее выполняются только задачи, запускаемые обработчиками прерываний (такими, как драйвер ввода с клавиатуры и часы реального времени), и операции, вызываемые транзитной программой из операционной системы. Система MS-DOS не обладает средствами поддержки режима разделения процессора между несколькими параллельно выполняемыми задачами и не может перехватить управление у программы при ее аварийном завершении или слишком большом времени выполнения. Такими возможностями обладает система MS OS/2, которая является системой защищенного режима с возможностью многозадачной работы с вытеснением (разделением времени).

ЗАГРУЗКА СИСТЕМЫ MS-DOS

При загрузке или перезагрузке системы выполнение начинается с адреса 0FFFF0H. Это определяется особенностями семейства микропроцессоров 8086/8088, а не самой системы MS-DOS. Системы, базирующиеся на этих процессорах, сконструированы так, что адрес 0FFFF0H лежит внутри области ПЗУ и содержит машинную команду передачи управления программе проверки системы и программе начального загрузчика ПЗУ (рис. 2.1).

Программа начального загрузчика ПЗУ считывает в некоторую область памяти программу дискового начального загрузчика, начиная с первого сектора загружаемого диска (сектор загрузчика), и затем передает ей управление (рис. 2.2). Кроме того, сектор загрузчика содержит таблицу с информацией о формате диска.

Программа дискового начального загрузчика проверяет наличие на диске копии системы MS-DOS. Для этого считывается первый сектор корневого каталога и определяется, являются ли первые два файла файлами IO.SYS и MSDOS.SYS (или IBMBIO.COM

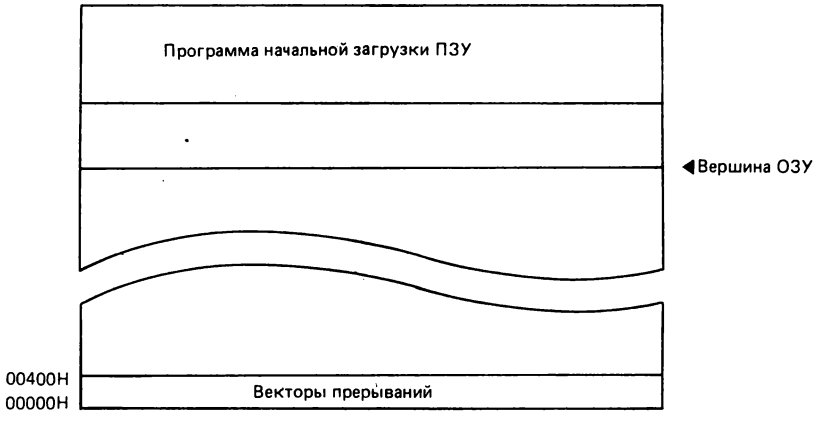


Рис. 2.1. Типичная компьютерная система на базе микропроцессоров 8086/8088 сразу после загрузки или перегрузки операционной системы.

Выполнение начинается с адреса 0FFFF0H, в котором содержится команда перехода, осуществляющая передачу управления программе начального загрузчика ПЗУ.

и IBMDOS.COM) (в указанном порядке). Если этих файлов там нет, то пользователю предлагается сменить диск и нажать на произвольную клавишу для новой попытки. Если оба системных файла найдены, то дисковый начальный загрузчик считывает их в память и передает управление на точку входа модуля IO.SYS (рис. 2.3). (В некоторых реализациях дисковый начальный загрузчик считывает в память только файл IO.SYS, который в свою очередь загружает файл MSDOS. SYS.)

Файл IO.SYS, загруженный с диска, обычно состоит из двух отдельных модулей.

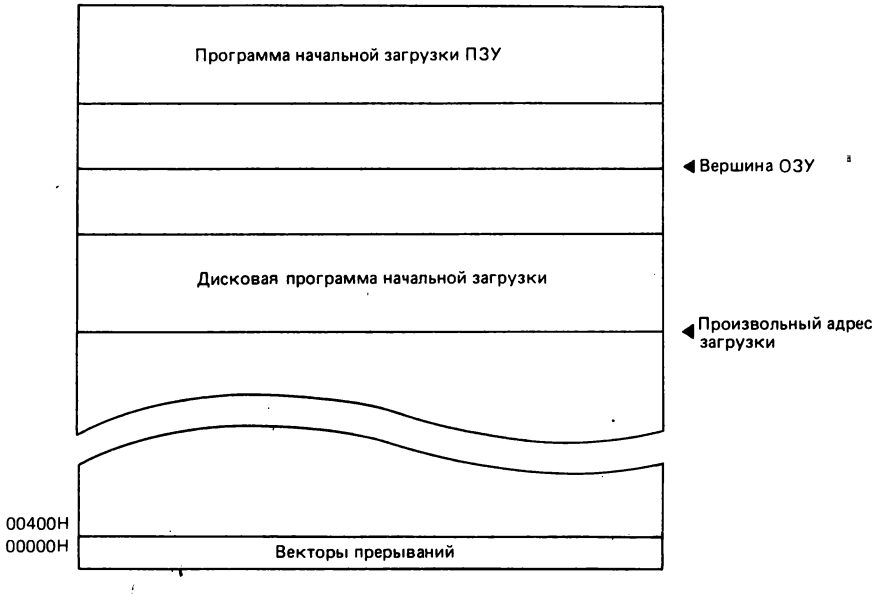


Рис. 2.2. Программа начальной загрузки ПЗУ считывает в память программу начальной загрузки диска, начиная с первого сектора загружаемого системного диска, и затем передает ей управление.

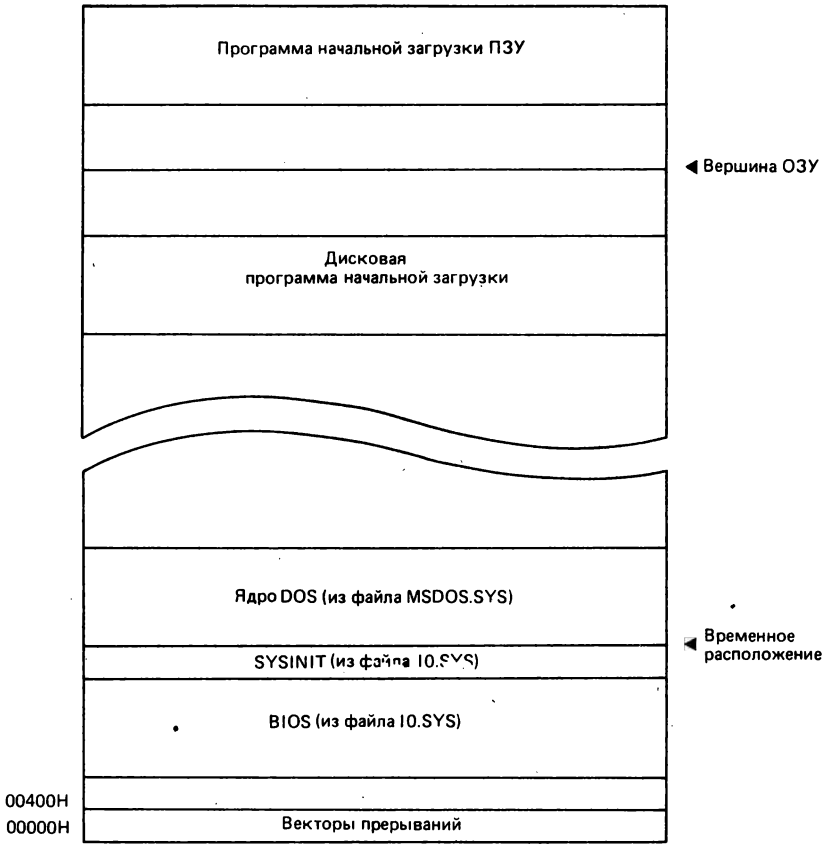


Рис. 2.3. Программа начальной загрузки диска считывет файл IO.SYS в память.

В этом файле находятся системные модули BIOS (драйверы резидентных устройств) и SYSINIT. Затем (в зависимости от типа BIOS) либо программа начального загрузчика диска, либо BIOS считывает из файла MSDOS.SYS в память ядро DOS.

Первый из них, BIOS, содержит связанный набор резидентных драйверов таких устройств, как консоль, последовательный порт, печатающее устройство, блочно-ориентированное устройство и таймер, а также программу инициализации, выполняемую только при начальной загрузке системы. Второй модуль, SYSINIT, поддерживаемый системой Microsoft, компонуется в файл IO.SYS, как и BIOS, изготовителем компьютера.

SYSINIT вызывается с помощью программы инициализации BIOS. Модуль определяет величину непрерывной памяти, доступной системе, и затем располагается по ее старшим адресам. Далее модуль переносит ядро системы DOS, MSDOS.SYS, из области ее начальной загрузки в область окончательного расположения в памяти, перекрывая начальную программу SYSINIT и любую другую расширяемую программу инициализации, которая содержалась в файле IO.SYS (рис. 2.4).

Далее SYSINIT вызывает программу инициализации в модуле MSDOS.SYS. Ядро DOS инициализирует ее внутренние таблицы и рабочие области, устанавливает векторы прерываний по адресам с 2OH по 2FH и перебирает связанный список

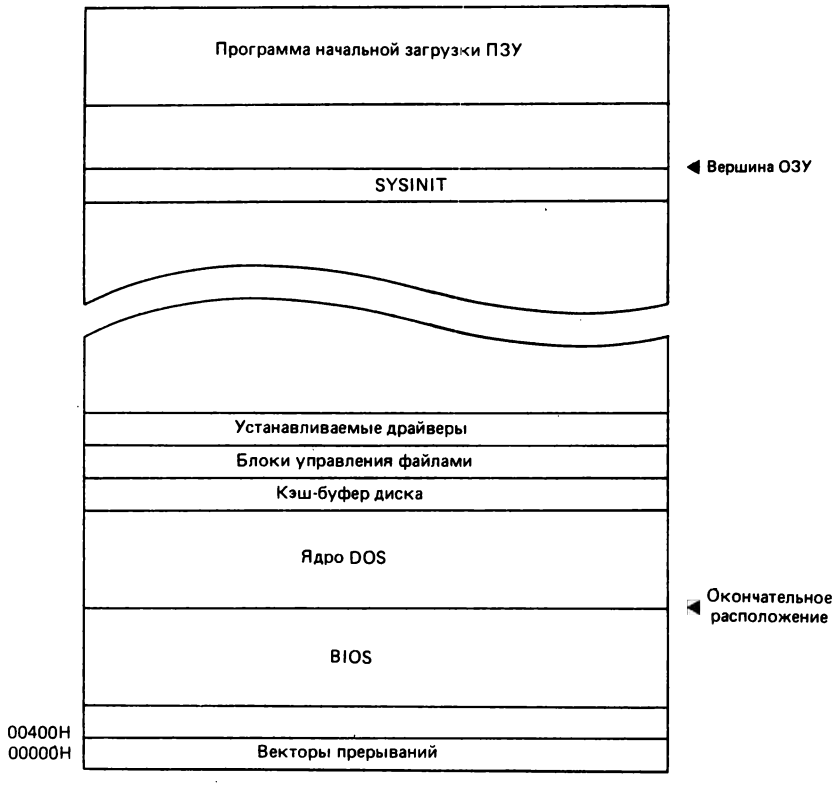


Рис. 2.4. Модуль SYSINIT перещает себя к старшим адресам памяти и сдвигает ядро системы DOS, файл MSDOS.SYS, в сторону младших адресов памяти в область окончательного расположения.

Выделяется память под кэш-буфер системного диска и блок управления файлами, а затм загружаются и компонуются с системой заданные в файле CONFIG.SYS устанавливаемые драйверы устройств.

резидентных драйверов устройств, вызывая функцию инициализации для каждого из них (см. гл. 14).

Функции драйверов определяют состояние оборудования, выполняют всю необходимую инициализацию аппаратуры и устанавливают векторы для внешних аппаратных прерываний имеющихся драйверов.

В процессе инициализации ядро DOS проверяет блоки параметров диска, возвращаемые с помощью резидентных драйверов блочно-ориентированных устройств, определяет наибольший размер сектора, который будет использован в системе, строит блоки параметров дисковода и выделяет буфер сектора диска, затем управление передается модулю SYSINIT.

Когда ядро DOS проинициализировано и все резидентные драйверы доступны, модуль SYSINIT может вызвать обычный файловый сервис системы MS-DOS и открыть файл CONFIG.SYS. Этот необязательный файл содержит различные команды, которые позволяют пользователю настраивать среду MS-DOS. Например, пользователь может задать дополнительные драйверы аппаратных устройств, число буферов

диска, максимально возможное число одновременно открытых файлов и имя файла командного процессора (оболочки).

Если файл CONFIG.SYS найден, он загружается в память для выполнения. Все строчные символы преобразуются в заглавные, и файл построчно интерпретируется, что приводит к последовательному выполнению команд. Память распределяется под кэш-буфер диска и внутренние блоки управления файлами, которые используются системными запись-ориентированными функциями и функциями типа дескриптора (см. гл. 8). Драйверы, указанные в файле CONFIG.SYS, последовательно загружаются в память, активизируются с помощью вызовов соответствующих модулей инициализации и заносятся в связанный список драйверов. Функции инициализации каждого из них сообщают модулю SYSINIT размер памяти, отведенной под соответствующий драйвер.

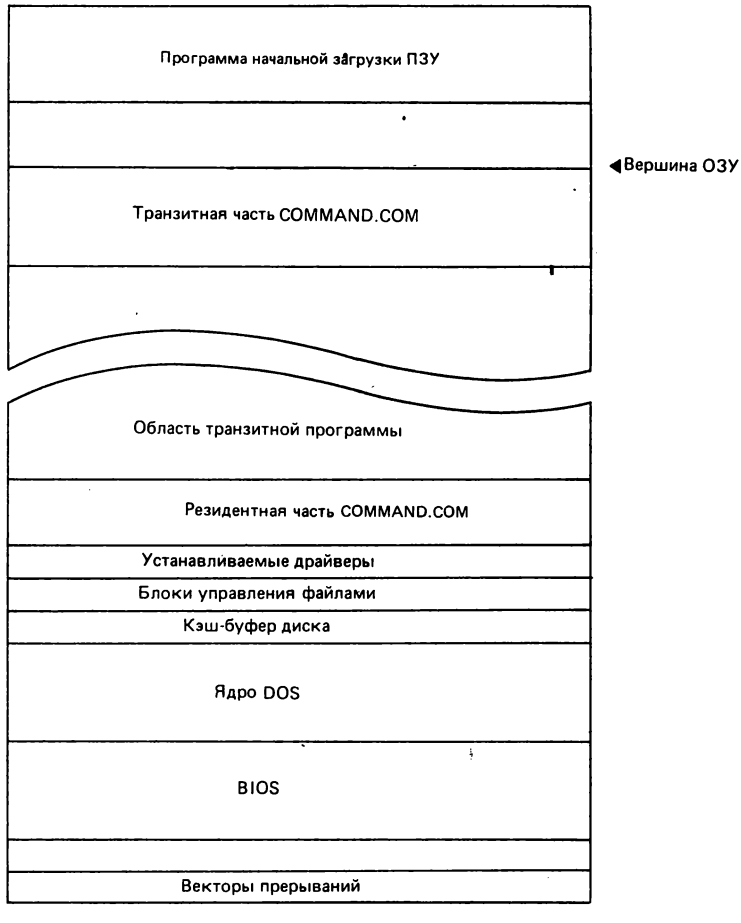


Рис. 2.5. Окончательный результат выполнения процедуры запуска системы MS-DOS в типичной вычислительной системе.

Резидентная часть COMMAND.COM размещается по младшим адресам памяти над ядром DOS. Транзитная часть, содержащая интерпретатор командных файлов и внутренние команды, помещается по старшим адресам памяти, где может перекрываться внешними командами системы и прикладными программами, которые исполняются в области транзитных программ.

После загрузки всех устанавливаемых драйверов SYSINIT закрывает все дескрипторы файлов и открывает вновь консоль (CON), принтер (PRN) и последовательный порт (AUX) как устройства стандартного ввода, стандартного вывода, стандартной ошибки, стандартной печати и стандартного последовательного устройства. Это позволяет символьно-ориентированному драйверу, установленному пользователем, замещать резидентные драйверы BIOS стандартных устройств.

В конце своего выполнения модуль SYSINIT вызывает системную функцию EXEC для загрузки интерпретатора командной строки, или оболочки. (Стандартной оболочкой является COMMAND.COM, однако с помощью файла CONFIG.SYS можно устанавливать и другие оболочки.) После загрузки оболочка высвечивает на экране дисплея запрос и ждет от пользователя ввода команды. Теперь система MS-DOS готова для работы и модуль SYSINIT уже не требуется (рис. 2.5).

ГЛАВА 3. СТРУКТУРА ПРИКЛАДНЫХ ПРОГРАММ В СРЕДЕ MS-DOS

Программы, которые выполняются под управлением системы MS-DOS, делятся на два основных типа: программы с расширением .COM, которые имеют максимальный размер приблизительно 64 Кбайт, и программы с расширением .EXE, которые могут иметь размер наибольшей доступной памяти. На языке микропроцессора Intel 8086 это означает, что программы типа .COM соответствуют минимальной модели, в которой все регистры сегмента содержат одну и ту же величину, т. е. программа и данные объединены. В противоположность этому программы типа .EXE образуют малую, среднюю или большую модели, в которых регистры сегмента содержат различные величины, т. е. программа, данные и стек постоянно находятся в отдельных сегментах. Программы типа .EXE могут иметь много сегментов программ и данных, адресация к которым осуществляется при помощи длинных вызовов и манипуляций с регистром сегмента данных DS.

Программа типа .COM постоянно хранится на диске в виде абсолютного образа памяти в файле с расширением .COM. У файла отсутствуют заголовок или какая-либо другая внутренняя служебная информация. С другой стороны, программа типа .EXE постоянно находится на диске в файле специального типа, который имеет индивидуальный заголовок, карту настройки, контрольную сумму и другую информацию, используемую (но не всегда) системой MS-DOS.

Оба типа программ с расширением .COM и .EXE переносятся в память для выполнения с помощью одного и того же механизма, а именно функции EXEC, которая образует загрузчик системы MS-DOS. Функция EXEC может вызываться с именем файла программы, загружаемой COMMAND.COM (обычным интерпретатором командной строки системы MS-DOS) либо другой оболочкой, пользовательским интерфейсом или другой программой, которая была загружена раньше с помощью EXEC. Если в области транзитных программ достаточно свободной памяти, то EXEC выделяет блок памяти под новую программу, строит префикс сегмента программы (PSP) по его базовому адресу, а затем считывает программу в память непосредственно над PSP. В конце своей работы EXEC устанавливает регистры сегмента и стека и передает управление программе.

После активизации программы EXEC может выдавать адреса дополнительной информации, такой, как хвост команды, блоки управления файлами, контекстный блок. При необходимости эту информацию можно передавать другой программе. (В гл. 12 обсуждается процедура применения функции EXEC в пользовательских программах, приведены примеры.)

Программы с расширением .COM и .EXE часто называют транзитными программами. Транзитная программа на время своего выполнения «владеет» блоком выделенной ей памяти и имеет почти полный контроль над системными ресурсами. При завершении работы такой программы по любой причине – снятие операционной системой, успешное выполнение и выход в MS-DOS – выделенный ей блок памяти освобождается (отсюда термин «транзитный») и может быть использован следующей загружаемой программой.

ПРЕФИКС СЕГМЕНТА ПРОГРАММЫ

Полное понимание сути префикса сегмента программы необходимо для успешного программирования в среде системы MS-DOS.

MS-DOS отводит префиксу область в 256 байт в начале блока памяти, выделяемого транзитной программе. Префикс имеет несколько связей с MS-DOS, которые могут использоваться транзитной программой; кроме того, определенную информацию записывает в него MS-DOS как для собственных целей, так и для передачи транзитной программе, которая в случае необходимости может информацию использовать (рис. 3.1).

Еще в первых версиях системы MS-DOS префикс был сделан совместимым с областью управления, которая (в соответствии с требованиями заслуженной операционной системы CP/M фирмы Digital Research) размещалась под транзитными программами, что позволяло переносить программы в MS-DOS без больших логических изменений. Хотя с той поры MS-DOS заметно изменилась, структура префикса в значительной степени совпадает с его аналогом в системе CP/M. Например, в префиксе по относительному адресу 0000H находится команда передачи управления системному обработчику завершения процесса, который удаляет программу после ее выполнения и осуществляет окончательный выход. Точно также в префиксе по относительному адресу 0005H находится команда связи с диспетчером функций MS-DOS, который осуществляет дисковые операции, операции ввода-вывода через консоль и другое сервисное обслуживание по запросу транзитной программы. Таким образом, обращения к PSP: 0000 и PSP: 0005 эквивалентны вызовам CALL 0000 и CALL 0005 в системе CP/M. (Однако эти обращения не являются «одобренным» способом получения сервисного обслуживания.)

В слове префикса с относительным адресом 0002H находится сегментный адрес верхней границы блока памяти, выделенного транзитной программе. Программа может с помощью этого значения определить, следует ли запросить больше памяти для своей работы и имеется ли лишняя память, которую можно освободить для нужд других процессов.

Область префикса с адреса 000AH по адрес 0015H предназначена для предыдущего содержимого векторов прерываний обработчиков завершения Ctrl-C и критической ошибки. Если транзитная программа изменит эти векторы для своих целей, то после ее завершения система MS-DOS восстановит исходные значения векторов, сохраненные в префиксе.

Слово префикса с относительным адресом 002CH хранит сегментный адрес блока окружения, в котором находится несколько ASCIIZ-строк (последовательности символов в коде ASCII, заканчивающиеся нулевым байтом). Блок окружения наследуется от программы, которая вызывала функцию EXEC для загрузки выполняющейся в данный момент программы. В блоке находится такая информация, как текущий путь, используемый программами файла COMMAND.COM для поиска выполнимых программ, расположение на диске самого файла COMMAND.COM, формат запроса пользователя, который выдается COMMAND.COM.

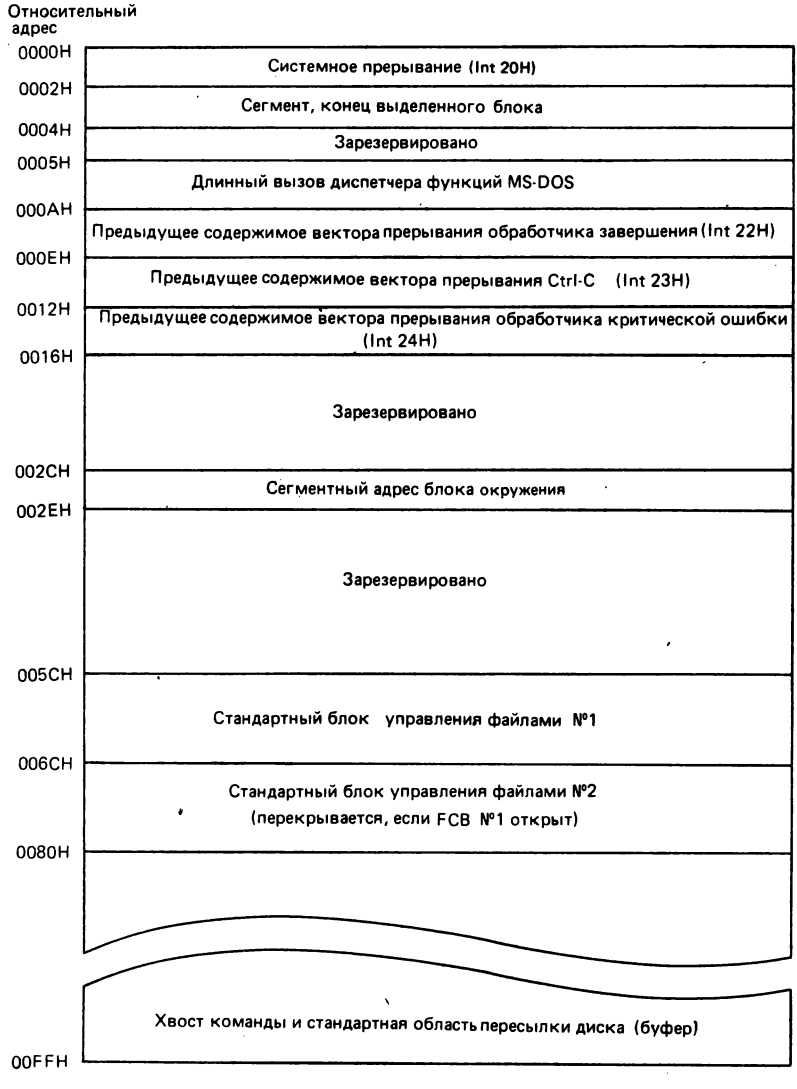


Рис. 3.1. Структура префикса сегмента программы.

Хвост команды – это остаток командной строки (после имени программы), запустившей транзитную программу; хвост команды копируется в префикс, начиная с относительного адреса 0081H. Длина хвоста команды, без учета символа возврата в конце строки, помещается в байт с относительным адресом 0080H. Параметры перенаправления или конвейеризации и связанные с ними имена файлов не встречаются в той части командной строки (хвосте команды), которая передается транзитной программе, так как операция переназначения прозрачна для прикладных программ.

Для того чтобы обеспечить совместимость с системной CP/M, MS-DOS выполняет выделение и перенос двух первых параметров хвоста команды в два стандартных блока управления файлами (file control block, FCB), расположенных в префиксе по адресам PSP:005CH и PSP:006CH, предполагая при этом, что оба параметра – имена файлов.

Однако если параметры – имена файлов с описанием пути, то в этих стандартных блоках управления файлами допустимым будет только код дисковода, так как функции типа FCB, с помощью которых осуществляется доступ к файлам и записям, не поддерживают иерархические файловые структуры. Прежде, когда совместимость с CP/M имела большее значение, стандартные блоки управления файлами были полезны, но в прикладных программах современной системы MS-DOS, которые должны поддерживать полный путь поиска, они в значительной степени бесполезны. (Блоки управления файлами подробно рассматриваются в гл. 8, а иерархические файловые структуры – в гл. 9.)

Область префикса длиной 128 байт от 0080H до 00FFH, устанавливаемая системой MS-DOS до передачи управления транзитной программе, служит также стандартной *дисковой областью пересылки (*disk transfer area, DTA). Если программа явным образом не именяет DTA, то все операции чтения или записи файла, затребованные вызовами группы функций типа FCB, автоматически используют эту область в качестве буфера данных. Это, впрочем, редко оказывается полезным, представляя еще один вариант работы с PSP, сохраненный в MS-DOS только ради совместимости с CP/M.

Предупреждение. Программы не должны изменять части префикса, расположенные ниже относительного адреса 005CH.

ВВЕДЕНИЕ В ПРОГРАММЫ ТИПА .COM

Программы типа .COM хранятся на диске в виде файлов, в которых содержится абсолютный образ машинных команд, предназначенных для выполнения. Так как файлы не содержат никакой настроечной информации, то они компактнее эквивалентных файлов типа .EXE, к тому же загружаются для выполнения немного быстрее. Заметим, что MS-DOS не пытается выяснить, действительно ли файл типа .COM содержит выполнимую программу (здесь в отличие от файла типа .EXE нет признака, ни контрольной суммы). Система заносит любой файл с расширением .COM в память и передает ему управление.

Так как программы типа .COM загружаются непосредственно над префиксом сегмента программы и, кроме того, не имеют заголовка, который может задавать другую точку входа, то их начальный адрес всегда составляет 0100H, что определено размером префикса. Максимальная длина программы типа .COM составляет 65536 байт минус длина префикса (256 байт) и обязательное слово стека (2 байта).

Когда управление передается от MS-DOS программе типа .COM, все регистры указывают на префикс (рис. 3.2). В указатель стека (stack pointer, SP), если позволяет память, помещается число 0FFFFH, в противном случае – максимальный адрес памяти минус 2 байта. (Система MS-DOS при входе в программу помещает в стек нулевое слово.)

Несмотря на то что размер выполнимого файла типа .COM не может превзойти 64 Кбайт, существующие в настоящее время версии системы MS-DOS выделяют программам типа .COM в момент их загрузки всю область транзитных программ. Так как такие программы часто являются наследием раннего периода MS-DOS и необязательно корректны в отношении управления памятью, то операционная система делает наихудшее предположение и дает программам типа .COM все, что возможно. Если программа типа .COM хочет использовать для возбуждения другого процесса функцию EXEC, то сначала она должна сократить выделяемую ей память до минимума, обеспечивающего продолжение ее работы, обращая внимание при этом на сохранение стека программы. (Этот вопрос подробно рассматривается в гл. 12.)

Когда программа типа .COM завершает выполнение, то вернуть управление системе MS-DOS она может несколькими способами. Системное прерывание Int 21H

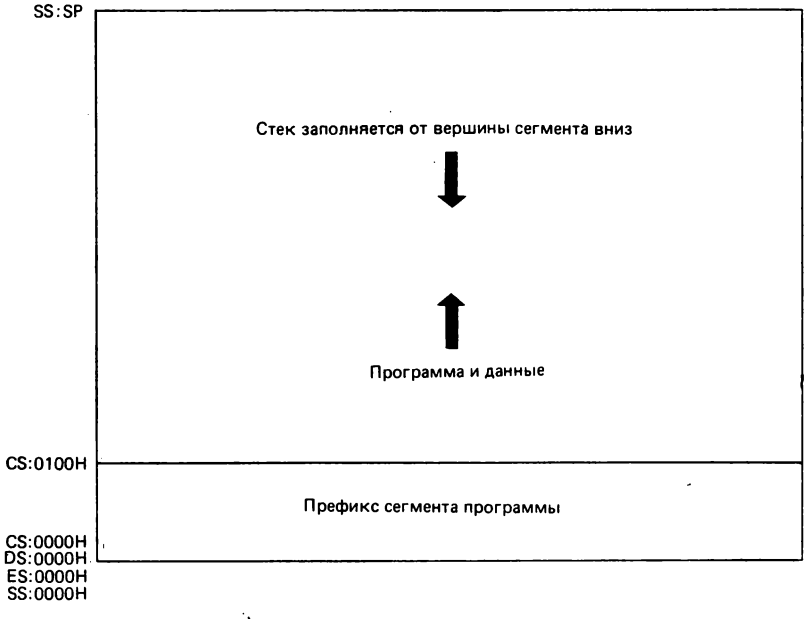


Рис. 3.2. Образ памяти типичной программы типа .COM сразу после загрузки.

Содержимое файла типа .COM заносится в память непосредственно над префиксом сегмента программы. Программа и данные расположены вместе в одном сегменте, и все регистры сегмента содержат одно и то же значение.

с функцией 4CH – это предпочтительный способ, позволяющий программе возвращать код завершения в ту программу, или командный процессор, или командный файл, где это прерывание было возбуждено. Если, однако, программа выполняется под управлением системы MS-DOS версии 1, то она должна завершаться с помощью прерываний Int 20H, Int 21H с функцией 0 или команды ближнего возврата NEAR RETURN. (Так как при входе в программу в стек было помещено слово ноль, то NEAR RETURN вызывает передачу управления по адресу PSP:0000, где находится команда прерывания Int 20H.)

Прикладная программа типа .COM может компоноваться из многих отдельных объектных модулей. Все модули должны использовать одно и то же имя сегмента программы и имя класса, причем модуль с точкой входа по относительному адресу 0100H внутри сегмента при компоновке должен быть первым. Кроме того, внутри программы типа .COM все процедуры должны иметь атрибут NEAR (ближний), так как вся выполнимая программа постоянно находится в одном сегменте.

В процессе компоновки программы типа .COM компоновщик выдает на экран дисплея сообщение

Warning: no stack segment (Предупреждение: нет сегмента стека)

Это сообщение можно игнорировать. Результатом работы компоновщика является файл типа .EXE, который до выполнения должен быть преобразован с помощью системной утилиты EXE2BIN в файл типа .COM. Затем вы можете удалить файл типа .EXE. (В гл. 4 дан пример такой операции.)

Пример программы типа .COM

На рис. 3.3 представлена структура простой программы на языке ассемблера HELLO.COM, которой предстоит превратиться в файл типа .COM. (Полезно сравнить листинги этой и другой программы, HELLO.EXE, приведенной в этой же главе ниже.) Так как эта программа очень короткая и простая, то относительно большую часть ее исходного текста занимают директивы ассемблера, которые не влияют на размеры создаваемого выполнимого модуля.

Оператор NAME (имя) определяет имя модуля, используемое в операции компоновки. Это помогает ориентироваться в карте, образуемой компоновщиком. В трансляторе MASM версии 5.0 и старше имя модуля всегда совпадает с именем файла, и оператор NAME игнорируется.

Команда PAGE (страница), когда она применяется с двумя операндами, как в строке 2 нашего примера, определяет длину и ширину страницы. Действующие по умолчанию значения – 66 строк и 80 столбцов в строке. Если вы применяете команду PAGE без операндов, то на принтер посылается сигнал перевода строки и печатается заголовок. В больших программах почаще используйте команду PAGE с тем, чтобы каждую вашу подпрограмму расположить на отдельной странице, что упростит чтение текста.

Команда TITLE (заголовок), строка 3, задает текст (не более 60 символов), который будет печататься в верхнем левом углу каждой страницы. Команда TITLE необязательна и не должна использоваться более одного раза в файле с исходным текстом.

Далее, пропустив несколько строк комментария и операторы EQU, мы приходим к объявлению сегмента программы, который начинается со строки 22 командой SEGMENT и кончается на строке 49 командой ENDS. Метка в самом левом поле строки 2 дает имя _TEXT сегменту программы. Поля операндов в правой части строки дают сегменту атрибуты WORD, PUBLIC и 'CODE'. (По-видимому, полезно почитать руководство по макроассемблеру фирмы Microsoft для детального знакомства с возможными атрибутами сегмента.)

Так как программа будет преобразована в файл типа .COM, то ее выполнимые коды и области данных будут находиться внутри одного сегмента программы. Кроме того, программа должна иметь в качестве начального относительный адрес 0100H (непосредственно над префиксом сегмента программы), что задается оператором ORG в строке 24.

Вслед за командой ORG в строке 27 мы видим оператор ASSUME. Концепция ASSUME часто является камнем преткновения для программистов, начинающих писать на языке ассемблера. В известном смысле ASSUME ничего не «выполняет», он только сообщает транслятору, какие сегментные регистры вы намереваетесь использовать для указания на сегменты вашей программы. Таким образом, транслятор может обеспечить замену сегментов, когда это необходимо. Важно подчеркнуть, что оператор ASSUME не следит за загрузкой регистров сегмента требуемыми значениями; он только объявляет транслятору о вашем намерении выполнить это в программе. (Помните, что в случае программы типа .COM MS-DOS инициализирует все регистры сегмента до входа в программу, так что они указывают на префикс.)

В сегменте программы мы встречаем еще один тип «блочного» объявления, которое начинается на строке 29 с команды PROC и закрывается на строке 40 командой ENDP. Эти две команды объявляют начало и конец процедуры, т. е. блока выполнимой программы, который осуществляет только одну функцию. Метка в левом поле оператора PROC (в данном случае print) определяет имя процедуры. Поле операнда дает процедуре атрибут. Если процедура имеет атрибут NEAR, то вызвать ее может только программа из того же сегмента, тогда как если атрибут процедуры FAR

```
 1:          name    hello
 2:          page    55,132
 3:          title   HELLO.COM-- Вывод 'HELLO' на терминал
 4:  ;
 5:  ;
 6:  ;HELLO.COM:  Демонстрация различных компонент
 7:  ;            функциональной программы типа  .COM
 8:  ;            на языке ассемблера и обращений к
 9:  ;            функциям MS-DOS.
10:  ;
11:  ; Рей Данкан, Май 1988
12:  ;
13:
14:  stdin   equ     0               ; Дескриптор стандартного ввода
15:  stdout  equ     1               ; Дескриптор стандартного вывода
16:  stderr  equ     2               ; Дескриптор стандартной ошибки
17:
18:  cr      equ     0dh             ; Код ASCII возврата каретки
19:  lf      equ     0ah             ; Код ASCII перевода строки
20:
21:
22:  _TEXT   segment word public 'CODE'
23:
24:          org     100h            ; У файлов типа .COM всегда
25:                                  ; начальный адрес 100h.
26:
27:          assume  cs:_TEXT,ds:_TEXT,es:_TEXT,ss:_TEXT
28:
29:  print   proc    near            ; Точка входа из MS-DOS
30:
31:          mov     ah,40h          ; Функция 40h = запись
32:          mov     bx,stdout       ; Дескриптор стандартного вывода
33:          mov     cx,msg_len      ; Длина сообщения
34:          mov     dx,offset msg   ; Адрес сообщения
35:          int     21h             ; Переход в MS-DOS
36:
37:          mov     ax,4c00h        ; Выход, код завершения = 0
38:          int     21h             ; Переход в MS-DOS
39:
40:  print   endp
41:
42:
43:  msg     db      cr,lf           ; Вывод сообщения на экран дисплея
44:          db      'Hello World!',cr,lf
45:
46:  msg_len equ     $-msg           ; Длина сообщения
47:
48:
49:  _TEXT   ends
50:
51:  end     print                   ; Определение точки входа
```

Рис. 3.3. Листинг программы HELLO.COM.

(дальний), то ее может вызвать программа, расположенная в любом месте пространства памяти, адресуемой центральным процессором (ЦП). В программах типа .COM все процедуры имеют атрибут NEAR.

Ради ясности примера я ограничился смехотворно простой процедурой print. Она вызывает прерывание MS-DOS Int 21H с функцией 40H для пересылки на экран

сообщения Hello World! и прерывание Int 21H с функцией 4CH для завершения программы.

Оператор END в строке 51 сообщает транслятору, что достигнут конец исходного файла и, кроме того, задает точку входа программы. Если относительный адрес точки входа отличается от 0100H, то файл типа .EXE, образующийся после трансляции и компоновки исходной программы, нельзя преобразовать в файл типа .COM.

ВВЕДЕНИЕ В ПРОГРАММЫ ТИПА .EXE

Мы только что рассмотрели программу, написанную таким образом, что ее можно оттранслировать и превратить в файл типа .COM. Структура такой программы очень проста, а поэтому программист, которому нужно скомпилировать подобного сорта быструю утилиту, может сосредоточиться на логике программы и в минимальной степени вникать в проблемы управления транслятором. Однако программы типа .COM имеют определенные недостатки, и поэтому наиболее серьезные программы на языке ассемблера в системе MS-DOS написаны так, что их можно преобразовать в файлы типа .EXE.

В то время как программы типа .COM имеют существенное ограничение на общий размер (не более 64 Кбайт в сумме на собственно программу, данные и стек), программы типа .EXE могут быть практически неограниченного размера (в пределах доступной памяти компьютера). Хотя обычный программный загрузчик в системе MS-DOS не использует этого достоинства файлов типа .EXE, но способность загружать отдельные части больших программ в разные участки памяти, а также благоприятная

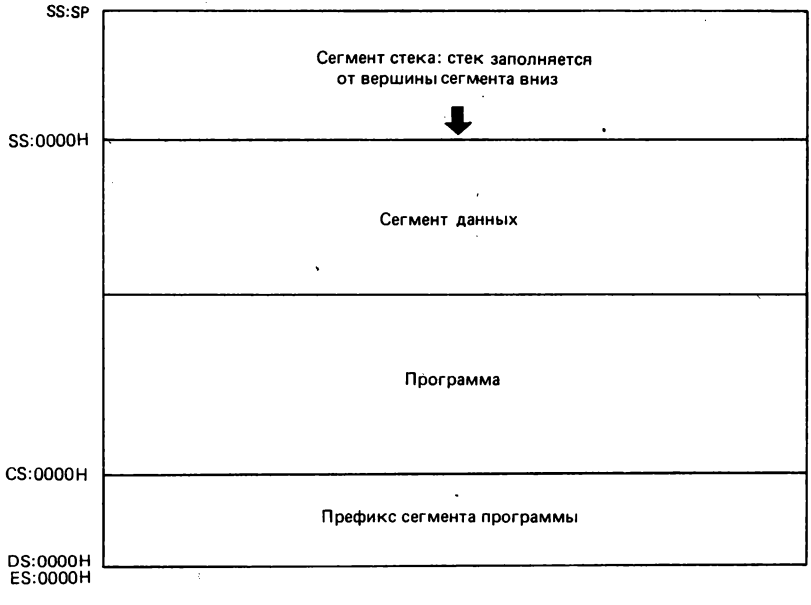


Рис. 3.4. Образ памяти типичной программы типа .EXE сразу после загрузки.

Файл типа .EXE настраивается и заносится в память над префиксом сегмента программы. Программа, данные и стек хранятся в отдельных сегментах, причем необязательно в том порядке, который приведен здесь. Точка входа может находиться в любом месте сегмента программы; она задается в операторе END главного модуля программы. Когда программа получает управление, регистры DS (сегменты данных) и ES (дополнительного сегмента) указывают на префикс сегмента программы. Программа обычно сохраняет это значение, а затем сбрасывает содержимое регистров DS и ES, настраивая их на область данных.

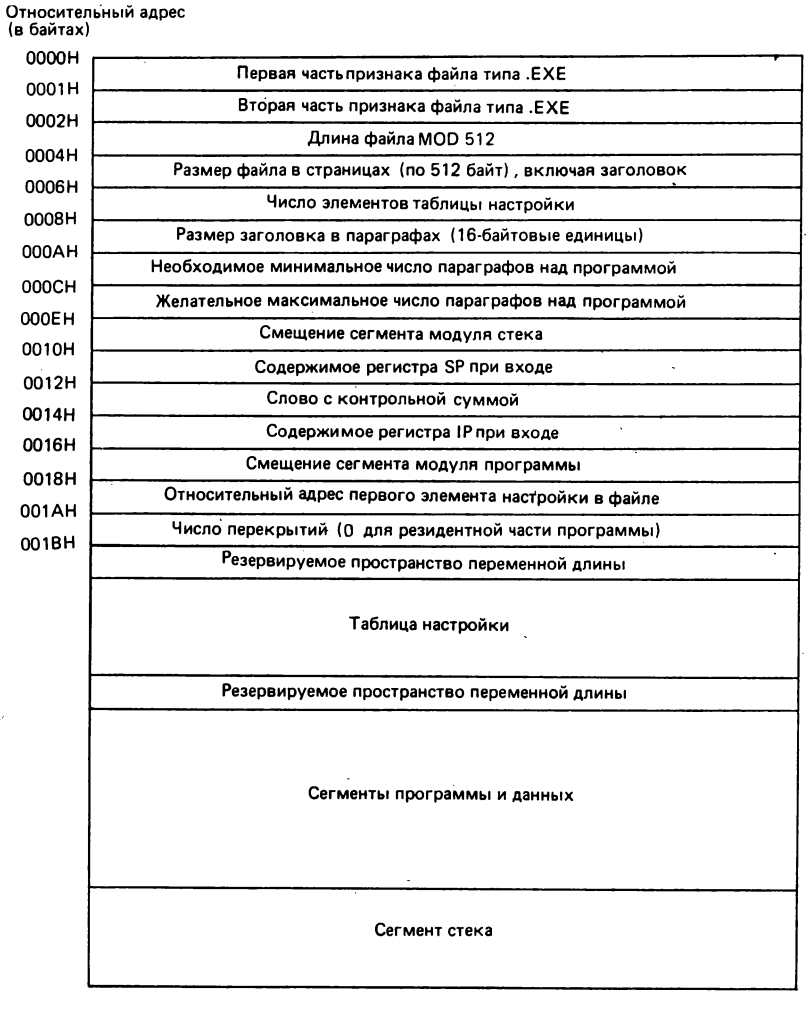


Рис. 3.5. Формат загрузочного модуля типа .EXE.

возможность выделить «чистый код» вашей программы [1], который может разделяться несколькими задачами,– все это чрезвычайно существенно в многозадачных средах типа Microsoft Windows.

В системе MS-DOS загрузчик всегда вводит программу типа .EXE в память непосредственно над префиксом сегмента программы, хотя порядок сегментов программы, данных и стека может меняться (рис. 3.4). Файл типа .EXE содержит заголовок, или блок управляющей информации характерного формата (рис. 3.5 и 3.6). Размер заголовка определяется числом команд программы, настраиваемых во время загрузки, но всегда кратен 512 байт.

До того как система MS-DOS передает управление программе, вычисляются

[1] Часть программы в машинных кодах, которая не содержит изменяемых ячеек. – *Прим. перев.*

```
C>DUMP HELLO.EXE
       0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000   4D 5A 28 00 02 00 01 00 20 00 09 00 FF FF 03 00  MZ(..... .......
0010   80 00 20 05 00 00 00 00 1E 00 00 00 01 00 01 00  .. .............
0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0050   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
       .
       .
       .
0200   B8 01 00 8E D8 B4 40 BB 01 00 B9 10 00 90 BA 08  ......@.........
0210   00 CD 21 B8 00 4C CD 21 0D 0A 48 65 6C 6C 6F 20  ..!..L.!..Hello
0220   57 6F 72 6C 21 0D 0A                             World!..
```

Рис. 3.6. Шестнадцатеричный дамп содержимого загрузочного модуля исходной программы HELLO.EXE.

Отметим следующие величины, представляющие интерес: в байтах 0000H и 0001H – признак типа .EXE, в байтах 0006H и 0007H – число элементов таблицы настройки, в байтах 000AH и 000BH – выделение минимальной дополнительной памяти (MIN_ALLOC), в байтах 000CH и 000DH – выделение максимальной дополнительной памяти (MAX_ALLOC), в байтах 0014H и 0015H – начальное содержимое регистра IP (указатель команд). Кроме того, см. рис. 3.5.

начальные значения регистра сегмента программы (code segment, CS) и указателя команд (instruction pointer, IP), при этом используются информация о точке входа из заголовка файла типа .EXE, а также адрес загрузки программы. Эта информация извлекается из оператора END исходного текста одного из модулей программы. Регистры сегмента данных (data segment, DS) и дополнительного сегмента (extra segment, ES) указывают на префикс, таким образом, программа может обращаться к указателю блока окружения, хвосту команды и другой полезной информации, находящейся в PSP.

Начальное содержимое регистра сегмента стека (stack segment, SS) и указателя стека (stack pointer, SP) устанавливается из заголовка. Эта информация берется из объявления сегмента с атрибутом STACK где-то в исходном тексте программы. Пространство памяти, выделяемое под стек, может инициализироваться тем или иным значением в зависимости от определения сегмента стека. Многие программисты предпочитают инициализировать пространство стека распознаваемой последовательностью данных, чтобы по дампу памяти можно было определить, какая часть стекового пространства действительно используется программой.

Когда программа типа .EXE завершает свою работу, то она должна вернуть управление системе MS-DOS с помощью прерывания Int 21H с функцией 4CH. Возможны другие методы, но у них нет никаких преимуществ, к тому же они менее удобны (так как обычно требуют, чтобы регистр CS указывал на PSP).

Программа типа .EXE, поступающая на вход компоновщика, может состоять из большого числа отдельных объектных модулей. Допускается, чтобы каждый модуль использовал уникальное имя сегмента программы, а процедуры имели атрибут либо NEAR, либо FAR в зависимости от условий определения имени и размеров выполнимой программы. Программист должен заботиться о том, чтобы компонуемые вместе модули содержали только один сегмент с атрибутом STACK и только одну точку входа, определяемую директивой ассемблера END. Результатом работы компоновщика является файл с расширением .EXE, который можно немедленно запустить на выполнение.

Пример программы типа .EXE

На рис. 3.7 представлена принципиальная структура программы на языке ассемблера HELLO.EXE, которой предстоит образовать файл типа .EXE. Как минимум, у программы должно быть имя модуля, сегмент программы, сегмент стека и основная процедура, которая после загрузки программы принимает от системы MS-DOS управление компьютером. Для придания примеру законченного вида в программе HELLO.EXE находится также сегмент данных.

Директивы NAME, TITLE и PAGE, рассмотренные в программе HELLO.COM, точно так же используются и здесь, поэтому мы не будем повторяться, а перейдем к следующим строкам программы. Пропустив несколько строк комментария и операторы EQU, мы приходим к объявлению сегмента программы, который начинается со строки 21 командой SEGMENT и кончается на строке 41 командой ENDS. Как и в программе HELLO.COM, метка в самом левом поле строки дает сегменту программы имя _TEXT. Поля операндов в правой части строки определяют атрибуты WORD, PUBLIC и 'CODE'.

Вслед за командой определения сегмента программы, на строке 23, мы находим оператор ASSUME. Отметим, что в отличие от программы HELLO.COM здесь в оператор ASSUME включены имена нескольких сегментов. Напомним еще раз, что этот оператор не оказывает прямого влияния на содержимое регистров сегмента, но влияет на действие самого транслятора.

Внутри сегмента программы на строке 25 командой PROC объявляется главная процедура print, которая закрывается на строке 39 командой ENDP. Поскольку процедура входит в файл типа .EXE, мы дали ей атрибут FAR, хотя в действительности в этом нет смысла, так как программа мала и процедура не вызывается из какой-либо точки той же программы.

Сначала, согласно указанию в операторе ASSUME, процедура print инициализирует регистр DS, загружая его значением, указывающим на базу области данных. (Система MS-DOS устанавливает регистры CS и SS автоматически.) Затем процедура использует системное прерывание Int 21H с функцией 40H для вывода на экран сообщения Hello World! точно так же, как и в программе HELLO.COM. И наконец, в строках 36 и 37 процедура возвращается в систему MS-DOS по прерыванию Int 21H с функцией 4CH, передавая при этом нулевой код завершения, что по соглашению означает успешное выполнение действия.

В строках с 44 по 51 объявляется сегмент данных с именем _DATA, в котором находятся используемые программой переменные и константы. Если разные модули программы имеют несколько сегментов данных с одним и тем же именем, то компоновщик соберет их вместе и поместит в один физический сегмент памяти.

Строки с 54 по 58 организуют стек сегмента; команды PUSH и POP будут обращаться к этой области рабочей памяти. До того как система MS-DOS передаст управление программе типа .EXE, она устанавливает регистры SS и SP в соответствие с объявленным размером и положением сегмента стека. Конечно, позаботьтесь о выделении под стек пространства не меньше, чем максимальная глубина стека, достигаемая во время выполнения программы, плюс достаточное число слов для содержимого регистров, сохраняемых в стеке при вызовах функций MS-DOS. Переполнение стека может разрушить сегменты вашей программы и данных, что в свою очередь вызывет странное поведение вашей программы или даже полный отказ компьютера! Оператор END на строке 60 завершает нашу короткую программу HELLO.EXE, сообщая транслятору о достижении конца исходного файла и обеспечивая метку точки входа в программу из системы MS-DOS.

Отличия программ типа .COM и .EXE представлены на рис. 3.8.

```
 1:          name    hello
 2:          page    55,132
 3:          title   HELLO.EXE-- Вывод 'Hello' на терминал
 4:  ;
 5:  ;HELLO.EXE:   Демонстрация различных компонент
 6:  ;             функциональной программы на языке
 7:  ;             ассемблера типа .EXE  и обращений
 8:  ;             к функции MS-DOS
 9:  ;
10:  ; Рей Данкан, Май 1988
11:  ;
12:
13:  stdin   equ     0               ; Дескриптор стандартного ввода
14:  stdout  equ     1               ; Дескриптор стандартного вывода
15:  stderr  equ     2               ; Дескриптор стандартной ошибки
16:
17:  cr      equ     0dh             ; Код ASCII возврата каретки
18:  lf      equ     0ah             ; Код ASCII перевода строки
19:
20:
21:  _TEXT   segment word public 'CODE'
22:
23:          assume  cs:_TEXT,ds:_data,SS:STACK
24:
25:  print   proc    far             ; Точка входа из MS-DOS
26:
27:          mov     ax,_DATA        ; Сделаем наш сегмент данных
28:          mov     ds,ax           ; адресуемым...
29:
30:          mov     ah,40h          ; Функция 40h = запись
31:          mov     bx,stdout       ; Дескриптор стандартного вывода
32:          mov     cx,msg_len      ; Длина сообщения
33:          mov     dx,offset msg   ; Адрес сообщения
34:          int     21h             ; Переход в MS-DOS
35:
36:          mov     ax,4c00h        ; Выход, код завершения = 0
37:          int     21h             ; Переход в MS-DOS
38:
39:  print   endp
40:
41:  _TEXT   ends
42:
43:
44:  _DATA   segment word public 'DATA'
45:
46:  msg     db      cr,lf           ; Вывод сообщения на экран дисплея
47:          db      'Hello World!',cr,lf
48:
49:  msg_len equ     $-msg           ; Длина сообщения
50:
51:  _DATA   ends
52:
53:
54:  STACK   segment para stack 'STACK'
55:
56:          db      128 dup (?)
57:
58:  STACK   ends
59:
60:  end     print                   ; Определение точки входа
```

Рис. 3.7. Листинг программы HELLO.EXE.

	Программа типа .COM	Программа типа .EXE
Максимальный размер (в байтах)	65536 минус 256 (префикс) минус 2 (стек)	Неограничен
Точка входа	PSP:0100H	Определяется оператором END
AL при входе	00H, если стандартный FCB № 1 содержит допустимый идентификатор дисковода, OFFH – недопустимый идентификатор дисковода	То же
AH при входе	00H, если стандартный FCB № 2 содержит допустимый идентификатор дисковода, OFFH – недопустимый идентификатор дисковода	" "
CS при входе	PSP	Сегмент, содержащий модуль с точкой входа
IP при входе	0100H	Относительный адрес точки входа программного сегмента
DS при входе	PSP	PSP
ES при входе	PSP	PSP
SS при входе	PSP	Сегмент с атрибутом STACK
SP при входе	min (0FFFEH, адрес вершины доступной памяти)	Размер стека, определяемый атрибутом STACK
Содержимое стека при входе	Слово со значением 0	Инициализирован или неинициализирован
Размер стека (в байтах)	65536 минус 256 (префикс) минус (размер выполнимой программы и данных)	Определяется в сегменте с атрибутом STACK
Обращение к подпрограмме	Обычно NEAR	NEAR или FAR
Способ завершения	Предпочтительнее командой Int 21H с функцией 4CH, в среде MS-DOS версии 1 командой NEAR RET	Предпочтительнее командой Int 21H с функцией 4CH
Размер файла	Размер программы	Размер программы плюс заголовок (кратный 512 байт)

Рис. 3.8. Основные различия между программами типа .COM и .EXE, включая необходимые условия их вызова.

ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О ПРОГРАММАХ НА ЯЗЫКЕ АССЕМБЛЕРА

Теперь, когда мы разобрали примеры работы программ на языке ассемблера типа .COM и .EXE, вернемся к ним еще раз и более формально обсудим их составные части. Все дальнейшее рассмотрение строится на базе макроассемблера фирмы Microsoft, который мы будем называть сокращенно MASM (the Microsoft Macro Assembler). Если вы хорошо владеете MASM и имеете опыт в написании программ на языке ассемблера, то при чтении книги можете пропустить этот раздел.

Программы на языке ассемблера MASM имеют три структурных уровня:

- Уровень модуля.
- Уровень сегмента.
- Уровень процедуры.

Модули – это части исходной программы, которые хранятся и транслируются независимо друг от друга. Сегменты – это физически сгруппированные подобные элементы (программные строки или данные); сегменты позволяют разделить группы разнородных данных. Процедуры – это функциональные подразделы выполнимой программы, т. е. подпрограммы, служащие для выполнения конкретных задач.

Модули программы

В системе MS-DOS структура уровня модуля состоит из файлов, содержащих исходные тексты отдельных программ. Каждый исходный файл преобразуется транслятором в перемещаемый объектный модуль. Объектный модуль может находиться автономно в отдельном файле, а также вместе с другими объектными модулями – в библиотеке объектных модулей, включающей часто используемые или взаимосвязанные подпрограммы. Компоновщик фирмы Microsoft (LINK) объединяет файлы объектных модулей в один файл выполнимой программы, добавляя к ним при необходимости модули из объектной библиотеки.

Применение модулей и библиотек объектных модулей приводит к уменьшению размеров файлов ваших исходных программ (и существенно увеличивает вашу производительность), так как в эти файлы уже не нужно включать исходные тексты подпрограмм, используемых совместно с другими программами. Кроме того, такая методика упрощает поддержку аппарата подпрограмм, так как необходимые изменения вносятся только в одну копию их исходных текстов, хранящуюся в одном месте, вместо могих копий в различных прикладных программах. В тех случаях, когда вы модернизируете одну из подпрограмм (или исправляете в ней ошибки), вам достаточно выполнить заново ее ассемблирование, поместить новый объектный модуль в библиотеку на место прежнего, еще раз скомпоновать все программы, использующие эту подпрограмму и voilà: во мгновение ока готов новый вариант.

Сегменты программы

Термин сегмент относится к двум разным понятиям программирования: физические сегменты и логические сегменты.

Физические сегменты – это блоки памяти по 64 Кбайт каждый. Микропроцессоры Intel 8086/8088 и 80286 включают четыре сегментных регистра, служащие исключительно в качестве указателей этих блоков. (Микропроцессор 80386 имеет шесть сегментных регистров, которые являются надмножеством набора регистров микропроцессоров 8086/8088 и 80286.) Каждый сегментный регистр указывает на нижнюю границу определенной области памяти размером 64 Кбайт. Таким образом, соответствующей манипуляцией сегментных регистров программа может адресоваться к любой ячейке памяти, но каждый раз в пределах 256 Кбайт.

Как мы уже видели, программы типа .COM предполагают, что все четыре сегментных регистра указывают на нижню границу программы. Тем самым ограничивается максимальный размер программы 64 Кбайт. С другой стороны, у программ типа .EXE есть возможность адресоваться ко многим физическим сегментам, а также переустанавливать по мере необходимости регистры сегментов с тем, чтобы они указывали на требуемые сегменты. Все последующие примеры книги посвящены программам типа .EXE.

Логические сегменты – это составные части программы. В любой программе типа

.EXE следует объявлять, как минимум, три логических сегмента: сегмент программы, сегмент данных и сегмент стека. Программы, в которых размеры исходного текста и данных превышают 64К, занимают больше, чем один сегмент программы или данных. Подпрограммы и данные, которые используются чаще, помещаются в целях повышения быстродействия в первичные сегменты программ или данных; используемые реже – во вторичные.

Объявляются сегменты с помощью директив SEGMENT и ENDS следующим образом:

```
ИМЯ     SEGMENT атрибуты

ИМЯ     ENDS
```

Атрибуты сегмента включают следующие его типы: выравнивания (align) – (BYTE, WORD или PARA), сочетания – (PUBLIC, PRIVATE, COMMON или STACK) и класса. Атрибуты сегмента учитываются компоновщиком, когда он объединяет логические сегменты для создания физического сегмента выполнимой программы. Практически вы можете прекрасно обходиться небольшим типичным набором атрибутов. Однако если вы захотите применять все имеющиеся атрибуты, то вам необходимо познакомиться с их описанием по соответствующему руководству.

Принадлежность программы к той или иной модели памяти основывается на числе сегментов программы и данных. В программах на языке ассемблера чаще всего применяется малая модель памяти, которая содержит по одному сегменту программы и данных, но кроме этого можно применять среднюю, компактную и большую модели памяти (рис. 3.9). (Имеются две дополнительные модели, которых в дальнейшем мы не будем касаться: минимальная модель, в которой в одном сегменте объединены программа и данные, например файл типа .COM в среде MS-DOS; максимальная модель, которая поддерживается оптимизирующим компилятором с языка Си фирмы Microsoft и позволяет применять структуры данных размером более 64 Кбайт.)

Фирма Microsoft установила для каждой модели памяти определенные имена сегмента и класса, которые используются всеми фирменными компиляторами с языков высокого уровня (рис. 3.10). Поскольку имена сегментов могут выбираться произвольно, то лучше применять имена, установленные фирмой Microsoft. Это упростит включение ваших подпрограмм, написанных на языке ассемблера, в программы, написанные на языках типа Си, а также применение в ваших программах (на языке ассемблера) библиотечных подпрограмм, составленных на языках высокого уровня.

Модель	**Количество сегментов программы**	**Количество сегментов данных**
Малая	Один	Один
Средняя	Много	Один
Компактная	Один	Много
Большая	Много	Много

Рис. 3.9. Модели памяти, обычно используемые в программах на языках ассемблера и СИ.

Модель памяти	Имя сегмента	Тип выравнивания	Тип сочетания	Тип класса	Группа
Малая	_TEXT	WORD	PUBLIC	CODE	
	_DATA	WORD	PUBLIC	DATA	DGROUP
	STACK	PARA	STACK	STACK	DGROUP
Средняя	Модуль _TEXT	WORD	PUBLIC	CODE	
	_DATA	WORD	PUBLIC	DATA	DGROUP
	STACK	PARA	STACK	STACK	DGROUP
Компактная	_TEXT	WORD	PUBLIC	CODE	
	данные	PARA	PRIVATE	FAR_DATA	
	_DATA	WORD	PUBLIC	DATA	DGROUP
	STACK	PARA	STACK	STACK	DGROUP
Большая	Модуль _TEXT	WORD	PUBLIC	CODE	
	данные	PARA	PRIVATE	FAR_DATA	
	_DATA	WORD	PUBLIC	DATA	DGROUP
	STACK	PARA	STACK	STACK	DGROUP

Рис. 3.10. Сегменты, группы и классы стандартных моделей памяти, используемые в программах на языке ассемблера.

Оптимизирующий компилятор с языка Си фирмой Microsoft и другие компиляторы с языков высокого уровня используют надмножество этих же сегментов и классов.

Другое важное соглашение, используемое в языках высокого уровня фирмы Microsoft, состоит в применении директивы GROUP, которая объявляет ближний сегмент данных (адресуемый программой относительно базы в регистре DS) и сегмент стека членами группы, имя которой DGROUP (автоматическая группа данных) распознается и компоновщиком, и программными загрузчиками систем Microsoft Windows и Microsoft OS/2. Директива GROUP приводит к объединению логических сегментов с различными именами в единый физический сегмент, так что к ним можно обращаться с помощью одного базового адреса сегмента. В программах на языке Си автоматическая группа данных (DGROUP) содержит также локальный пул (heap), который используется исполняющей системой Си для динамического выделения небольших областей памяти.

Для программ на языке ассемблера, которые будут выполняться под управлением MS-DOS, вы можете игнорировать DGROUP. Однако если вы планируете объединять эти программы с подпрограммами на языках высокого уровня, то должны следовать соглашению DGROUP фирмы Microsoft. Если, например, вы хотите включить подпрограммы из библиотеки программ Си в программу на языке ассемблера, то в начале вашей программы должна быть строка

```
DGROUP group _DATA,STACK
```

Последнее соглашение фирмы Microsoft, касающееся создания программ типа .EXE,– это порядок сегментов. Компиляторы языков высокого уровня работают в предположении, что первыми всегда идут сегменты программы, за ними следуют дальние сегменты данных, затем ближние сегменты данных и в конце – стек и пул. Этот порядок вам будет безразличен до тех пор, пока вы не начнете объединять программу, написанную на языке ассемблера, с библиотечными подпрограммами на языках высокого уровня, однако лучше с самого начала следовать данному соглашению.

Программные процедуры

Уровень процедуры как единицы структуры программы частично реален, частично умозрителен. По существу процедура – это что-то вроде маскарадной маски для подпрограммы.

В программе процедуры объявляются директивами PROC и ENDP следующим образом:

```
имя     PROC атрибут

        RET
имя
        ENDP
```

Атрибут NEAR или FAR, назначенный в определении PROC, указывает ассемблеру, какой тип вызова вы предполагаете использовать для входа в процедуру, т. е. будет ли процедура вызываться из других программ того же сегмента или программ, расположенных в других сегментах. Когда ассемблер встречает в процедуре команду RET, он

использует информацию атрибута, для того чтобы образовать правильный код операции либо для ближнего (внутрисегментного), либо для дальнего (межсегментного) возврата.

В любой программе должна быть главная процедура, которая принимает управление от системы MS-DOS. Включая имя главной процедуры в оператор END одного из исходных файлов программы, вы задаете точку входа в программу. На самом деле атрибут процедуры (NEAR или FAR) не слишком важен, так как программа возвращает управление системе MS-DOS не с помощью команды RET, а через вызов функции. Однако по соглашению большинство программистов присваивают главной процедуре атрибут FAR.

Разбивать оставшуюся часть программы на процедуры нужно аккуратным образом, так, чтобы каждая процедура осуществляла тщательно определенную элементарную функцию, возвращала результат процедуры в место ее вызова и избегала действий, приводящих к глобальным эффектам в программе. В идеале процедуры активизируют друг друга только с помощью команд CALL, имеют по одной точке входа и выхода, а завершаются с помощью команды RET, а не путем перехода на какие-то ячейки в той же программе.

Для простоты понимания и обслуживания процедуры необходимо, чтобы ее размер не превышал одной страницы (60 строк). Если процедура занимает больше чем одну страницу, то скорее всего она слишком сложна и вы должны передать какую-нибудь ее функцию одной или нескольким вспомогательным процедурам. Вы должны снабдить исходный текст каждой программы подробным комментарием, который фиксирует последовательность вызова процедуры, возвращаемые результаты, используемые регистры, а также все доступные или модифицируемые элементы данных. Усилия, потраченные на то, чтобы сделать ваши процедуры компактными, чистыми, гибкими и хорошо документированными, будут многократно вознаграждены при использовании ваших процедур в других программах.

ГЛАВА 4. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПРОГРАММИРОВАНИЯ В СИСТЕМЕ MS-DOS

Подготовка новой программы к выполнению под управлением системы MS-DOS – процесс итерационный, состоящий из четырех основных шагов:

- Работа с текстовым редактором для создания или модификации файла исходной программы в коде ASCII.
- Работа с ассемблером или компилятором языка высокого уровня (например, Macro Assembler фирмы Microsoft или C Optimizing Compiler той же фирмы) для трансляции исходного файла и получения перемещаемой объектной программы.
- Работа с компоновщиком для преобразования перемещаемой объектной программы в выполнимый в среде MS-DOS загрузочный модуль.
- Работа с отладчиком в целях проверки логики и поиска ошибок в программе.

Возможно, программисту покажется полезным или даже необходимым включить сюда следующие дополнительные утилиты программного обеспечения системы MS-DOS:

- LIB – для создания и поддержки библиотек объектных модулей.
- CREF – для создания листинга перекрестных ссылок.
- EXE2BIN – для преобразования файлов типа .EXE в файлы типа .COM.
- MAKE – для сравнения даты создания файлов и выполнения на этой основе последующих действий.

В этой главе дается обзор инструментальных средств фирмы Microsoft для программирования в системе MS-DOS, который включает ассемблер, компилятор с языка Си,

компоновщик и библиотекарь. В целом приведенная здесь информация относится и к средствам программирования фирмы IBM, которые в действительности представляют собой продукты фирмы Microsoft с незначительными изменениями и другими номерами версий. Даже если вы не программируете на языках Си или ассемблера, вам необходимо иметь хотя бы представление об этом инструментарии, так как все примеры в справочниках по системе DOS фирм IBM и Microsoft написаны на одном из этих языков.

Обзор в этой главе вместе с примерами программ и справочными данными в этой же книге должны дать опытному программисту достаточно информации, чтобы он немедленно начал писать ценные программы. Читатели, не обладающие достаточной подготовкой в области языков программирования Си и ассемблера или архитектуры микропроцессоров Intel 80/86, должны обратиться к обучающим и справочным пособиям, указанным в конце этой главы.

ТИПЫ ФАЙЛОВ

Инструментальные средства программирования в системе MS-DOS позволяют создавать и обрабатывать файлы разных типов. По соглашению у файлов могут быть следующие расширения:

Расширение	Тип файла
.ASM	Файл исходной программы на языке ассемблера
.C	Файл исходной программы на языке Си
.COM	Загрузочный модуль, выполнимый в среде MS-DOS, которому при выполнении настройка не нужна
.CRF	Образуемый ассемблером файл с информацией о перекрестных ссылках; предназначен для последующей обработки утилитой CREF.EXE
.DEF	Файл определения модулей, описывающий поведение сегмента программы (только для программ в средах MS OS/2 и Microsoft Windows; к обычным прикладным программам среды MS-DOS не применяется)
.EXE	Загрузочный модуль, выполнимый в среде MS-DOS, которому при выполнении требуется настройка
.H	Для программ на языке Си файл заголовка, содержащий исходный текст с константами, макрокомандами и функциями; директивой #include объединяется с программой на языке Си
.INC	Файл включения для программ на языке ассемблера, обычно содержащий макрокоманды и/или определения системных величин, таких, как коды ошибок
.LIB	Файл библиотеки объектных модулей, составленный из одного или более .OBJ-файлов; просматривается и обрабатывается программой LIB.EXE
.LST	Образуемый ассемблером листинг программы, который включает распределение памяти, машинные коды, исходный текст программы и сообщения об ошибках
.MAP	Листинг идентификаторов и их распределение в загрузочном модуле; образуется компоновщиком
.OBJ	Образуемый ассемблером или компилятором файл перемещаемого объектного модуля исходной программы
.REF	Листинг перекрестных ссылок, образуемый программой CREF.EXE по данным из .CRF-файла

МАКРОАССЕМБЛЕР ФИРМЫ MICROSOFT

Дистрибутив макроассемблера фирмы Microsoft (MASM) – это файл MASM.EXE. В момент трансляции ему требуется следующая информация:

- Имя файла, содержащего исходную программу.
- Имя файла создаваемой объектной программы.
- Имя устройства, на которое направляется листинг трансляции.
- Имя файла для информации, которая позднее обрабатывается утилитой перекрестных ссылок (CREF.EXE).

Вызвать MASM можно двумя способами. Если вы введете только имя ассемблера, то транслятор запросит имена всех входных и выходных файлов. Как видно из следующего примера, транслятор может дать стандартные ответы на все запросы, кроме имени исходного файла:

```
C>MASM <Enter>

Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

Source filename [.ASM]: HELLO <Enter>
Object filename [HELLO.OBJ]: <Enter>
Source listing [NUL.LST]: <Enter>
Cross-reference [NUL.CRF]: <Enter>

  4900 6 Bytes symbol space free

       0 Warning Errors
       0 Severe Errors

C >
```

Во всех запросах MASM можно применять имя логического устройства (например, PRN или COM1) для выдачи результата трансляции не в файл, а на символьное устройство. Заметим, что для листинга трансляции и файлов перекрестных ссылок стандартным является устройство NUL, т.е. файлы не создаются. Если вы в конце ответа даете символ "точка с запятой", то транслятор полагает, что все остающиеся ответы должны быть стандартными.

Более эффективный способ работы с MASM требует указания всех параметров в командной строке

```
MASM [опции] исх_файл, [объект_файл], [листинг], [ссылки]
```

Например, следующие командные строки эквивалентны выполнению диалога пользователя с системой:

```
C>MASM HELLO,,NUL,NUL <Enter>
```

или

```
C>MASM HELLO; <Enter>
```

Ключ	Описание
/A	Расположить сегменты в алфавитном порядке
/Bn	Установить размер буфера исходного файла (в Кбайтах)
/C	Создать файл перекрестных ссылок (.CRF)
/D	Создавать листинг на обоих проходах (для выявления фазовых ошибок)
/D*symbol*	Определить symbol как имя пустой текстовой строки (на symbol можно ссылаться с помощью условных директив ассемблера, включенных в файл)
/E	Транслировать для эмулятора арифметического сопроцессора 80 × 87, используя формат действительного числа IEEE
/I*путь*	Установить путь поиска включения файлов
/L	Создать файл листинга программы
/LA	Включить в листинг все генерируемые коды
/ML	Установить чувствительность к регистру клавиатуры во всех именах (имена, написанные прописными буквами, отличаются от своих строчных эквивалентов)
/MX	Сохранить строчные буквы только во внешних именах (т. е. именах, определяемых директивами PUBLIC или EXTRN)
/MU	Преобразовать в именах все строчные буквы в прописные
/N	Подавить генерацию таблиц макрокоманд, структур, записей, сегментов, групп и идентификаторов в конце листинга
/P	Проверить наличие смешанного кода при работе в защищенном режиме процессора 80286/80386
/T	«Сжатый» режим; подавить все сообщения, за исключением ошибок, выявленных во время трансляции
/V	«Расширенный» режим сообщений; указать в конце трансляции число строк и идентификаторов
/Wn	Установить уровень отображения ошибок (предупреждений); n = 0 ÷ 2
/X	Вывести в листинг ложные предложения условной трансляции
/Z	Вывести на экран строки исходной программы, содержащие ошибки
/Zd	Включить в .OBJ-файл информацию о номерах строк
/Zi	Включить в .OBJ-файл информацию о номерах строк и символах

Рис. 4.1. Ключи макроассемблера версии 5.1 фирмы Microsoft.

Эти команды рассматривают файл HELLO.ASM в качестве исходного, генерируют файл объектного кода HELLO.OBJ и отправляют файлы листинга и перекрестных ссылок в «мусорную корзину» [1].

Транслятор MASM допускает присутствие в командной строке необязательных ключей для управления объектным модулем и выходными файлами. На рис. 4.1 перечислены ключи, допускаемые MASM версии 5.1. Как показано в приведенном ниже примере, вы имеете возможность помещать часто используемые необязательные параметры в переменную окружения MASM, где их автоматически найдет транслятор:

```
C>SET MASM=/T /Zi <Enter>
```

Ключи в переменной окружения будут замещены на те значения, которые вы введете в командной строке.

В других версиях макроассемблера фирмы Microsoft может допускаться больше или меньше дополнительных ключей. Для выяснения этого вопроса смотрите руководство по той версии MASM, с которой вы работаете.

MASM позволяет пользователю заменять действующие по умолчанию расширения любого файла. Это довольно опасное свойство. Например, если вы в рассмотренном

[1] Здесь подразумевается устройство NUL. – *Прим. перев.*

примере на вопрос об имени объектного файла ответите – HELLO.ASM, то транслятор воспримет ваши условия без всяких комментариев и уничтожит исходный файл. Конечно, маловероятно, что это случится при работе в интерактивном режиме, однако вы должны быть очень внимательны к расширениям файлов, если обращение к MASM включено в командный файл.

ОПТИМИЗИРУЮЩИЙ КОМПИЛЯТОР ЯЗЫКА СИ ФИРМЫ MICROSOFT

Оптимизирующий компилятор языка Си фирмы Microsoft состоит из трех выполнимых файлов: C1.EXE, C2.EXE и C3.EXE, которые реализуют Си-препроцессор, транслятор, генератор кодов и оптимизатор программы. Дополнительная управляющая программа CL.EXE выполняет по порядку три файла компилятора, передавая каждому из них необходимую информацию с именами файлов и опциями компиляции.

Прежде чем начать работу с Си-компилятором и компоновщиком, вам необходимо установить четыре контекстные переменные:

Переменная	Действие
PATH = путь	Задать местоположение трех выполнимых файлов Си-компилятора (C1, C2 и C3), если они находятся вне текущего каталога; используется программой CL.EXE
INCLUDE = путь	Задать местоположение файлов директивой # include (стандартное расширение .H), не найдены в текущем каталоге
LIB = путь	Задать местоположение (или местоположения) библиотек объектных модулей, которые не найдены в текущем каталоге
TMP = путь	Задать местоположение временных рабочих файлов, создаваемых Си-компилятором и компоновщиком

Программа CL.EXE не поддерживает интерактивный режим и файлы ответов. Вы должны активизировать ее командной строкой следующего содержания:

CL [опции] файл [файл...]

Вы можете указать любое число файлов, причем файл с расширением .C будет преобразован в файл перемещаемого объектного модуля (.OBJ). Обычно если компилятор не находит ошибок, то все результирующие .OBJ-файлы, а также дополнительные .OBJ-файлы, заданные в командной строке, автоматически передаются компоновщику вместе с именами соответствующих библиотек поддержки.

У Си-компилятора много необязательных ключей, которые управляют его моделями памяти, выходными файлами, генерацией кодов и оптимизацией программ, что представлено на рис. 4.2. Синтаксис Си-компилятора в основном взят из систем UNIX/XENIX, и не надо искать в нем какой-то особый смысл.

Ключ	Описание
/Ax	Установить модель памяти: C = компактная модель H = максимальная модель

Рис. 4.2. Ключи оптимизурующего компилятора версии 5.1 с языка Си фирмой Microsoft.

Ключ	Описание
	L = большая модель
	M = средняя модель
	S = малая модель (по умолчанию)
/c	Только компилировать; компоновщик не вызывать
/C	Комментарии не убирать
/D <имя> [= текст]	Определить макрокоманду
/E	Переслать выход препроцессора на стандартный выход
/EP	Переслать выход препроцессора на стандартный выход без нумерации строк
/F <n>	Установить размер стека (число байт в шестнадцатеричном представлении)
/Fa [имя_файла]	Генерировать листинг трансляции программы на языке ассемблера
/Fc [имя_файла]	Генерировать смешанный листинг исходной и объектной программ
/Fe [имя_файла]	Назначить имя выполнимому файлу
/Fl [имя_файла]	Генерировать листинг объектного модуля
/Fm [имя_файла]	Генерировать файл карты загрузки
/Fo [имя_файла]	Назначить имя файлу объектного модуля
/FPx	Установить тип операций с плавающей точкой:
	a = обращения к дополнительной математической библиотеке
	c = обращения к библиотеке эмулятора
	c87 = обращения к библиотеке сопроцессора 8087
	i = встроенные операции с эмулятором (по умолчанию)
	i87 = встроенные операции с сопроцессором 8087
/Fs [имя_файла]	Генерировать листинг исходной программы
/Gx	Установить генерацию команд:
	0 = команды процессора 8086 (по умолчанию)
	1 = команды процессора 186
	2 = команды процессора 286
	c = функциональные вызовы по типу языка Паскаль
	s = без проверки стека
	t[n] = порог размера данных
/H <n>	Задать длину внешнего имени
/I <путь>	Задать дополнительный путь для # include
/J	Задать по умолчанию тип char как беззнаковый
/link [опции]	Передать компоновщику ключи и библиотечные имена
/Ox	Установить тип оптимизации:
	a = игнорировать совмещение имен[1)]
	d = запретить оптимизацию
	i = разрешить внутренние функции
	l = разрешить оптимизацию циклов
	n = запретить «ненадежную» оптимизацию
	p = разрешить оптимизацию точности представления
	r = запретить встроенный возврат
	s = оптимизировать по занимаемому пространству памяти
	t = оптимизировать по скорости выполнения (по умолчанию)
	w = игнорировать совмещение имен, за исключением имен в функциях
/P	Переслать выход препроцессора в файл
/Sx	Установить управление листингом исходного текста:
	l <позиции> = установить ширину строки
	p <число_строк> = установить длину страницы
	s <строка> = установить подзаголовок
	t <строка> = установить заголовок

Рис. 4.2.

Ключ	Описание
/Tc <файл>	Компилировать файл без расширения. Си
/u	Удалить все ранее определенные макрокоманды
/U <имя>	Удалить указанные макрокоманды (из числа определенных ранее)
/V <строка>	Установить строку версии
/W <n>	Установить уровень предупреждений (0–3)
/X	Игнорировать «стандартные места» для включенных файлов
/Zx	Установить прочее управление компиляцией: a = запретить расширение c = сделать функции языка Паскаль нечувствительными к регистру клавиатуры d = включить информацию о номерах строк e = разрешить расширение (по умолчанию) g = генерировать объявление i = включить информацию для символического отладчика l = удалить данные библиотеки, действующей по умолчанию p <n> = паковать структуры по границе n байт s = проверять только синтаксис

[1] Т.е. ситуацию, при которой в некотором окружении один и тот же объект допущен под разными именами.– *Прим. пер.*

Рис. 4.2.

КОМПОНОВЩИК ОБЪЕКТНЫХ МОДУЛЕЙ ФИРМЫ MICROSOFT

Объектный модуль, образуемый из исходного файла ассемблером MASM, имеет перемещаемую информацию, кроме того, в нем могут быть неразрешенные ссылки к внешним ячейкам памяти или подпрограммам. Его формат общий и для объектных модулей, образуемых компиляторами с языков высокого уровня (типа Фортран и Си) и предназначенных для выполнения под управлением MS-DOS. Компьютер не может выполнять объектные модули без их дополнительной обработки.

Компоновщик the Microsoft Object Linker (LINK), поставляемый в виде файла LINK.EXE, принимает один или более объектных модулей, разрешает внешние ссылки, подсоединяет все необходимые программы из указанных библиотек, производит всю необходимую настройку смещений и записывает файл, загружаемый и выполнимый в среде MS-DOS. Результатом работы компоновщика LINK является создание загрузочного модуля типа .EXE (см. гл. 3).

Как и при работе с MASM, вы можете задавать компоновщику LINK его параметры интерактивно или, вводя всю требуемую информацию, в одной командной строке. Если вы введете только имя компоновщика, то состоится диалог такого типа:

```
C>LINK <Enter>

Microsoft (R) Overlay Linker Version 3.61
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

Object Modules [.OBJ]: HELLO <Enter>
Run File [HELLO.EXE]: <Enter>
List File [NUL.MAP]: HELLO <Enter>
```

Libraries [.LIB]: <Enter>

C >

Если вы работаете с LINK версии 4.0 или старше, то компоновщик также запросит имя файла определения модулей (.DEF). В ответ на такой запрос нажмите клавишу Enter. Файлы определения модулей применяются при создании системами Microsoft Windows или MS OS/2 «новых .EXE»-выполнимых файлов, которые, однако, неуместны в обычных прикладных программах MS-DOS.

В примере входным файлом был HELLO.OBJ; выходными файлами были HELLO.EXE (выполнимая программа) и HELLO.MAP (карта распределения памяти, выдаваемая компоновщиком после разрешения всех ссылок и адресаций). На рис. 4.3 изображена карта распределения памяти.

Вы получите тот же результат значительно быстрее, вводя все параметры в командной строке следующим образом:

LINK опции объектный_файл, [exe-файл], [map-файл], [библиотеки]

Таким образом, командная строка, эквивалентная приведенному выше примеру интерактивного ввода, имеет вид

C>LINK HELLO,HELLO,HELLO,, <Enter>

или

C>LINK HELLO,,HELLO; <Enter>

Если в конце командной строки ввести символ «точка с запятой», то LINK примет, что значения всех остальных параметров устанавливаются по умолчанию.

Третий способ управления программой LINK заключается в использовании файла ответов. Файл ответов состоит из строк текста, соответствующих вашим ответам

```
 Start  Stop   Length Name                   Class
(Начало Конец  Длина  Имя                    Класс)
 00000H 00017H 00018H _TEXT                  CODE
 00018H 00027H 00010H _DATA                  DATA
 00030H 000AFH 00080H STACK                  STACK
 000B0H 000BBH 0000CH $$TYPES                DEBTYP
 000C0H 000D6H 00017H $$SYMBOLS              DEBSYM

  Address         Publics by Name

  Address         Publics by Value
 ( Адрес )

 Program entry point at 0000:0000
(Точка входа в программу по адресу 0000:0000)
```

Рис. 4.3. Карта, образуемая компоновщиком объектных модулей (LINK) фирмы Microsoft в процессе генерации программы HELLO.EXE, представленной в гл. 3.

Программа содержит сегмент программы CODE, данных DATA и стека STACK. Первая выполнимая команда находится в первом байте сегмента программы CODE. Сегменты $$TYPES и $$SYMBOLS не являются частью программы, а содержат информацию для отладчика CodeView. Оба сегмента игнорируются обычным загрузчиком системы MS-DOS.

в процессе интерактивного вызова компоновщика. Вы задаете имя файла ответов в командной строке, начиная с символа @:

LINK @имя_файла

Вы также можете ввести имя файла в ответ на любой запрос. Если файл ответов определен не полностью, то LINK будет запрашивать у вас пропущенную информацию.

Вводя команды компоновщика, вы можете описывать много объектных файлов с помощью оператора + или пробелов, как показано в следующем примере:

C>LINK HELLO+VMODE+DOSINT,MYPROG,,GRAPHICS; <Enter>

Эта команда выполнит компоновку файлов HELLO.OBJ, VMODE.OBJ и DOSINT.OBJ с образованием файла MYPROG.EXE, при этом библиотечный файл GRAPHICS.LIB будет использован для разрешения всех ссылок на символы, не определенные в описанных объектных файлах. LINK использует текущие дисковод и каталог, если они не указаны явным образом в имени файла. Не следует ожидать, что автоматически будут использованы те же дисковод и каталог, которые вы описали для предшествующего файла в той же командной строке.

С помощью оператора + или символов пробела в поле библиотеки вы можете указать до 32 библиотечных файлов, в которых будет осуществляться поиск. Любой компилятор с языка высокого уровня знает имена библиотек, используемых по умолчанию. Эти библиотеки автоматически просматриваются во время процесса компиляции при условии, что компоновщик может их найти (если они явно не исключены по ключу /NOD). LINK ищет библиотеки сперва в текущем каталоге назначенного по умолчанию дисковода, затем – по путям, указанным в командной строке, и уже потом – вдоль пути или путей, описанных переменной LIB, если она присутствует в окружении.

LINK допускает несколько необязательных ключей в командной строке или в конце любого интерактивного запроса. Эти ключи перечислены на рис. 4.4. Число допустимых ключей и их действия варьируются в зависимости от версии LINK. Для получения детальной информации о вашей конкретной версии смотрите руководство по компоновщику объектных модулей фирмы Microsoft.

Ключ	Полный формат	Описание
/A:n	/ALIGNMENT:n	Установить фактор выравнивания сектора сегмента. N должно быть равным степени 2 (по умолчанию = 512). Не связано с выравниванием логического сегмента. Относится только к сегментированным выполнимым файлам (системы Microsoft Windows и OS/2)
/B	/BATCH	Подавить запрос компоновщика, если библиотека не может быть найдена в текущем каталоге или ячейках памяти, описанных переменной окружения LIB
/CO	/CODEVIEW	Включить информацию символического отладчика в .EXE-файл для использования программой CodeView

Рис. 4.4. Ключи компоновщика объектных модулей (LINK) версии 5.0 фирмы Microsoft.

Ранние версии загрузчика используют подмножество этих ключей. Отметим, что допускается сокращенное обозначение ключей, однозначно описывающее эти ключи.

Ключ	Полный формат	Описание
/CP	/CPARMAXALLOC	Установить поле в заголовке .EXE-файла, управляющее объемом памяти, выделяемой программе дополнительно к памяти, необходимой для программных кодов, стека и инициализированных данных
/DO	/DOSSEG	Использовать стандарты фирмы Microsoft на имена и порядок сегментов
/DS	/DSALLOCATE	Загрузить данные по старшим адресам сегмента данных. Относится только к программам реального режима
/E	/EXEPACK	Упаковывать выполнимый файл, убирая последовательности повторяющихся байтов, и оптимизировать таблицу настройки
/F	/FARCALLTRANSLATION	Оптимизировать с целью ускорения дальние обращения к меткам внутри одного физического сегмента, заменяя их ближними вызовами и командами NOP
/HE	/HELP	Вывести на экран дисплея информацию о допустимых опциях
/HI	/HIGH	Загрузить программу по самым старшим возможным адресам
/I	/INFORMATION	Выводить на экран дисплея промежуточную информацию о процессе компоновки, включая номера проходов и имена скомпилированных объектных файлов
/INC	/INCREMENTAL	Образовать файлы .SYM и .ILK для последующего использования программой ILINK (инкрементным компоновщиком). Может использоваться без ключа /EXEPACK. Относится только к выполнимым файлам с сегментной организацией (системы Microsoft Windows и MS OS/2)
/LI	/LINENUMBERS	Записать адрес первой команды каждой строки исходной программы в файл карты. Не исполняется, если компилятор не включает информацию о номере строки в объектный модуль. Создание файла карты
/M[:n]	/MAP[:n]	Создать .MAP-файл с перечислением всех общих идентификаторов, упорядоченных по именам и расположению. Необязательная величина n – максимальное число идентификаторов, которые могут быть упорядочены (по умолчанию n = 2048). Если n указано, то список с сортировкой по алфавиту опускается
/NOD	/NODEFAULTLIBRARY-SEARCH	Отменить просмотр любых библиотек компилятора, действующих по умолчанию, ссылки на которые имеются в .OBJ-файле
/NOE	/NOEXTENDEDDICT-SEARCH	Игнорировать расширенный словарь библиотеки, если он присутствует. Обычно расширенный словарь обеспечивает компоновщик информацией о внутримодульных зависимостях, что приводит к ускорению процесса компоновки
/NOF	/NOFARCALLTRANS-LATION	Запретить оптимизацию дальних обращений к меткам внутри одного сегмента
/NOG	/NOGROUPASSOCIATI-ON	Игнорировать групповые связи при присвоении адресов элементам данных и программы
/NOI	/NOIGNORECASE	Не игнорировать регистр клавиатуры в именах во время процесса компоновки

Рис. 4.4.

Ключ	Полный формат	Описание
/NON	/NONULLSDOSSEG	Расположить сегменты аналогично ключу /DOSSEG, но не включать 16 нулевых байт в начало сегмента _TEXT
/NOP	/NOPACKCODE	Не упаковывать непрерывные логические сегменты программы в один физический сегмент
/O:n	/OVERLAYINTERRUPT:n	Использовать номер прерывания n в диспетчере оверлеев, поставляемом вместе с некоторыми языками высокого уровня фирмы Microsoft
/PAC [:n]	/PACKCODE [:n]	Упаковать непрерывные логические сегменты программы в один физический сегмент программы. Необязательное значение n равно максимальному размеру каждого упакованного физического сегмента программы (по умолчанию = = 65 536 байт). Сегменты в различных группах не упаковываются
/PADC:n	/PADCODE:n	Добавить n символов-заполнителей в конец каждого программного модуля, с тем чтобы позднее с помощью программы ILINK можно было вставить модуль бóльшего размера. Относится только к выполнимым файлам с сегментной организацией (системы Windows и MS OS/2)
/PADD:n	/PADDATA:n	Добавить n символов-заполнителей в конец каждого модуля данных, с тем чтобы позднее с помощью программы ILINK можно было вставить модуль большего размера. Относится только к выполнимым файлам с сегментной организацией (системы Microsoft Windows и MS OS/2)
/PAU	/PAUSE	Сделать паузу во время процесса компоновки, давая возможность сменить диски до записи .EXE-файла
/SE:n	/SEGMENT:n	Установить максимальное число сегментов в скомпонованной программе (по умолчанию = 128)
/ST:n	/STACK:n	Установить размер стека программы в байтах; игнорировать размер сегмента стека, объявленный в объектных модулях и файле определения
/W	/WARNFIXUP	Выводить на экран дисплея предупреждающие сообщения об адресах относительно базового адреса сегмента, который отличается от базового адреса группы. Относится только к выполнимым файлам с сегментной организацией (системы Microsoft Windows и MS OS/2)

Рис. 4.4.

УТИЛИТА EXE2BIN

Утилита EXE2BIN (EXE2BIN.EXE) преобразует .EXE-файл, созданный с помощью LINK, в выполнимый .COM-файл, если программа удовлетворяет следующим предпосылкам:

- Она содержит не более одного объявленного сегмента и не определяет стек.
- Ее длина менее 64 Кбайт.
- Ее начальный адрес 0100H.
- Первая ячейка файла задается в исходной программе с помощью директивы END как точка входа.

Хотя .COM-файлы несколько компактнее .EXE-файлов, следует избегать их использования. Программы с раздельными сегментами для команд, данных и стека

значительно проще переносятся в операционную среду типа MS OS/2 с защищенным режимом, а кроме того, .COM-файлы не поддерживают символическую отладочную информацию, требующуюся для программы CodeView.

Еще одно применение утилиты EXE2BIN связано с преобразованием настраиваемого драйвера устройства (после его трансляции и компоновки в .EXE-файл) в файл образа памяти типа .BIN или .SYS с начальным адресом ноль. Это преобразование необходимо в системе MS-DOS версии 2, в которой нельзя загружать драйверы в виде .EXE-файлов. Процесс разработки настраиваемого драйвера обсуждается в деталях в гл. 14. В отличие от других программных утилит EXE2BIN не обладает интерактивным режимом. Она всегда берет исходные и выходные имена файлов, разделенные пробелом, из командной строки MS-DOS следующим образом:

EXE2BIN исходный_файл [выходной_файл]

Если расширение исходного файла не указано, то по умолчанию принимается .EXE, а расширение выходного файла – .BIN. Если имя выходного файла не указано, то EXE2BIN дает ему имя входного файла с расширением .BIN.

Например, для того, чтобы преобразовать файл HELLO.EXE в HELLO.COM, следует применить такую командную строку:

C> EXE2BIN HELLO.EXE HELLO.COM <Enter>

Кроме того, программа EXE2BIN имеет и другие возможности, как, например, чисто двоичное преобразование с фиксированным размещением сегмента в целях создания программных образов для размещения в ПЗУ. Однако ввиду того, что эти особенности редко используются в прикладных программах для MS-DOS, они здесь не будут обсуждаться.

Утилита CREF

Утилита перекрестных ссылок (cross-reference, CREF), составляющая файл CREF.EXE, обрабатывает .CRF-файл, порождаемый MASM, и создает текстовый файл в кодах ASCII со стандартным расширением .REF. Этот файл содержит листинг перекрестных ссылок всех объявленных в программе идентификаторов и номера строк, где они встречаются (рис. 4.5). Такой листинг очень полезен в процессе отладки большой программы на языке ассемблера, содержащей много взаимозависимых процедур и переменных.

Свои параметры CREF может получать в диалоговом режиме или в командной строке. При вводе только имени утилиты CREF запросит имена входного и выходного файлов, как показано в следующем примере:

```
C> CREF <Enter>

Microsoft (R) Cross-Reference Utility Version 5.10
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.
Cross-reference [.CRF]: HELLO <Enter>
Listing [HELLO.REF]:
15 Symbols

C>
```

```
Microsoft Cross-Reference  Version 5.10       Thu May 26 11:09:34 1988
HELLO.EXE --- Вывод 'Hello' на терминал
  Symboll Cross-Reference     (# definition,  + modification)Cref-1
(Таблица перекрестных ссылок) (# определение, + модификация )

@CPU . . . . . .  1#
@VERSION . . . . . .  1#

CODE . . . . . .  21
CR . . . . . .  17#  46   47

DATA . . . . . .  44

LF . . . . . .  18#  46   47

MSG. . . . . .  33   46#
MSG_LEN. . . . . .  32   49#

PRINT. . . . . .  25#  39   60

STACK. . . . . .  23   54#  54   58
STDERR . . . . . .  15#
STDIN. . . . . .  13#
STDOUT . . . . . .  14#  31

_DATA. . . . . .  23   27   44#  51
_TEXT. . . . . .  21#  23  41

 15 Symbols
```

Рис. 4.5. Листинг перекрестных ссылок HELLO.REF, образованный утилитой CREF на основе файла HELLO.CRF для программы HELLO.EXE (пример из гл. 3).

Идентификаторы, объявленные в программе, представлены в алфавитном порядке слева. Против каждого идентификатора справа указаны номера строк, где он встречается. Число со знаком # означает номер строки, где данный идентификатор объявляется. Числа со знаком + определяют номер строк, где данный идентификатор модифицируется. Номера строк, данные в листинге перекрестных ссылок, совпадают с номерами строк, образуемыми транслятором с языка ассемблера в файле листинга программы (.LST), и не совпадают с нумерацией строк в файле исходного текста программы.

Кроме того, параметры можно ввести в командной строке в следующем виде:

CREF CRF_файл, листинг_файла

Например, следующая программная строка эквивалентна приведенному выше диалогу:

C> CREF HELLO,HELLO <Enter>

Если утилита CREF не может найти заданный .CRF-файл, то на экране дисплея выводится сообщение об ошибке. В противном случае листинг перекрестных ссылок передается в заданный файл на диске. С помощью команды COPY можно направить файл на принтер:

COPY ЛИСТИНГ_ФАЙЛА PRN:

Листинг перекрестных ссылок можно направить непосредственно на символьное устройство, указав в ответ на запрос Listing имя устройства.

БИБЛИОТЕЧНАЯ СИСТЕМА ФИРМЫ MICROSOFT

Хотя объектные модули, создаваемые программой MASM или компиляторами с языков высокого уровня, можно компоновать прямо в выполнимые загрузочные модули, возможно их помещение в специальные файлы – библиотеки объектных модулей. В библиотеке модули отыскиваются по именам или по содержащимся в них идентификаторам с описанием PUBLIC, что дает возможность компоновщику извлекать требуемый модуль в процессе разрешения внешних ссылок в программе.

Библиотекарь the Microsoft Library Manager (LIB), поставляемый в виде файла LIB.EXE, создает и поддерживает программы библиотеки, добавляя, корректируя и уничтожая по мере необходимости объектные файлы. Кроме того, LIB может проверить внутреннюю непротиворечивость библиотечного файла или распечатать таблицу его содержимого (рис. 4.6).

LIB соблюдает соглашения о командах большинства программных средств фирмы Microsoft. Вы должны предоставить библиотекарю имя рабочего библиотечного файла, описать выполнимые операции (одну или несколько), указать имя листинга файла или устройства и (необязательно) имя выходной библиотеки. Если вы не укажете имя выходной библиотеки, то LIB даст ей имя входной библиотеки и изменит расширение последней на .BAK.

Операции LIB задаются просто в виде имен объектных файлов, предваряемых

```
_abort.....abort          _abs.....abs
_access.....access        _asctime.....asctime
_atof.....atof            _atoi.....atoi
_atol.....atol            _bdos.....bdos
_brk.....brk              _brkctl.....brkctl
_bsearch.....bsearch      _calloc.....calloc
_cgets.....cgets          _chdir.....dir
_chmod.....chmod          _chsize.....chsize
     .
     .
     .

_exit          Offset: 00000010H  Code and data size: 44H
             (относительный адрес) (размер программы и данных)
  __exit

_filbuf        Offset: 00000160H  Code and data size: BBH
  __filbuf

_file          Offset: 00000300H  Code and data size: CAH
  __iob          __iob2             __lastiob
     .
     .
     .
```

Рис. 4.6. Фрагмент листинга библиотечного файла SLIBC.LIB программ на языке Си фирмы Microsoft, образованного библиотекарем the Microsoft Library Manager (LIB).

Первая часть листинга – алфавитный список всех имен с атрибутивом PUBLIC, объявленных во всех модулях библиотеки. Каждое имя связывается с объектным модулем, которому оно принадлежит. Вторая часть листинга – алфавитный список имен объектных модулей из библиотеки вместе с их относительными адресами внутри библиотечного файла и общими размерами модулей (в байтах). Запись, описывающая модуль, включает в себя также список всех имен с атрибутом PUBLIC, объявленных внутри него.

префиксным символом, который описывает требуемое действие:

Префикс	Значение
–	Удалить объектный модуль из библиотеки
*	Извлечь модуль и поместить его в отдельный .OBJ-файл
+	Добавить в библиотеку объектный модуль или все содержимое другой библиотеки

Вы можете комбинировать командные префиксы. Например, комбинация – + приводит к замещению модуля, а сочетание *– к извлечению модуля в новый файл с последующим удалением его из библиотеки.

В том случае, когда вы вызываете LIB, указав только имя, библиотекарь запрашивает всю необходимую информацию в интерактивном режиме, как это демонстрируется в следующем примере:

```
C> LIB <Enter>

Microsoft (R) Library Manager Version 3.08
Copyright (C) Microsoft Corp 1983-1987. All rights reserved.

Library name: SLIBC <Enter>
Operations: +VIDEO <Enter>
List file: SLIBC.LST <Enter>
Output library: SLIBC2 <Enter>

C>
```

В этом примере программа LIB добавила объектный модуль VIDEO.OBJ в библиотеку SLIBC.LIB, записала оглавление библиотеки в файл SLIBC.LST и назвала созданную новую библиотеку SLIBC2.LIB.

Библиотекарь может вызываться и командной строкой следующего вида:

```
LIB библиотека [команды], [список], [новая_библиотека]
```

Например, следующая командная строка эквивалентна приведенному выше диалогу:

```
C> LIB SLIBC+VIDEO,SLIBC.LST,SLIBC2; <Enter>
```

Как и в других утилитах фирмы Microsoft, символ «точка с запятой» в конце командной строки заставляет LIB использовать для всех пропущенных параметров ответы, действующие по умолчанию.

Подобно LINK, программа LIB может также принимать команды из файла ответов. Содержимое файла составляют строки текста, в точности соответствующие ответам, которые вы давали бы LIB в диалоге. Вы описываете имя файла ответа в командной строке, предварив его символом @ следующим образом:

```
LIB @имя_файла
```

LIB имеет только три ключа: /I(IGNORECASE), /N(NOIGNORECASE) и /RAGESIZE:number.

Ключ /IGNORECASE действует по умолчанию. Ключ /NOIGNORECASE заставля-

ет LIB различать идентификаторы, которые отличаются лишь видом (прописные или строчные) составляющих их букв. Значение ключа /PAGESIZE, определяющего величину единицы изменения выделяемого для данной библиотеки пространства, следует задавать сразу же после имени файла библиотеки. Размер страницы библиотеки определяется в байтах и должен быть равен степени двойки в пределах от 16 до 32 768 (16, 32, 64 и т. д.). По умолчанию размер страницы равен 16 байт. Так как каталог библиотеки всегда имеет фиксированное число страниц, то устанавливая большой размер страницы, вы сможете хранить больше объектных модулей в библиотеке, но, с другой стороны, это приведет к большему неиспользуемому пространству внутри файла.

УТИЛИТА MAKE

Утилита MAKE (MAKE.EXE) сравнивает даты создания или последней модификации файлов и выполняет команды, основываясь на результатах такого сравнения. Благодаря этой одной, но довольно существенной возможности утилита MAKE может быть полезна для поддержки сложных программ, построенных из большого числа модулей. Даты исходного, объектного и выполнимого файлов сравниваются в логическом порядке. Ассемблер, компилятор, компоновщик и другие инструментальные средства программирования вызываются по мере необходимости.

Утилита MAKE обрабатывает открытый текстовый файл в коде ASCII, называемый файлом действия. Вы запускаете утилиту, вводя командную строку следующего вида:

MAKE файл_действия [опции]

По установленному соглашению файл действия имеет такое же имя, как и обрабатываемый выполнимый файл, но без расширения. Возможные ключи утилиты MAKE перечислены на рис. 4.7.

Обычный файл действия состоит из одного или нескольких операторов зависимости, разделенных пустыми строками. Любой оператор зависимости может следовать за списком команд системы MS-DOS следующим образом:

выходной_файл:исходный_файл.

    команда

    команда

Если дата и время какого-нибудь исходного файла превышают соответствующие параметры выходного файла, то выполняются команды, указанные в перечне. В файле действия можно в любом количестве использовать строки с комментарием, которые начинаются с символа #. Кроме того, утилита MAKE обрабатывает операторы перенаправления и макроопределения. Подробности этих более тонких возможностей можно найти в документах фирмы Microsoft или IBM.

Ключ	Описание
/D	Вывести на экран по мере обработки файлов дату их последней модификации
/I	Игнорировать коды завершения (возврата), выдаваемые командами и программами, выполненными как следствие предложений зависимости
/N	Вывести на экран команды, которые должны выполняться как следствие предложений зависимости, но не выполнять эти команды
/S	Не выводить на экран команды по мере их выполнения
/X <имя_файла>	Направлять сообщения об ошибках из утилиты MAKE или любой другой программы, исполняющей утилиту MAKE, в указанный файл. Если в качестве имени файла используется дефис (-), то сообщения об ошибках направляются на устройство стандартного вывода

Рис. 4.7. Ключи утилиты MAKE.

КОМПЛЕКСНЫЙ ПРИМЕР

Давайте объединим все полученные нами сведения об использовании инструментальных средств программирования в среде MS-DOS. На рис. 4.8 изображена блок-схема всех этапов построения выполнимой программы.

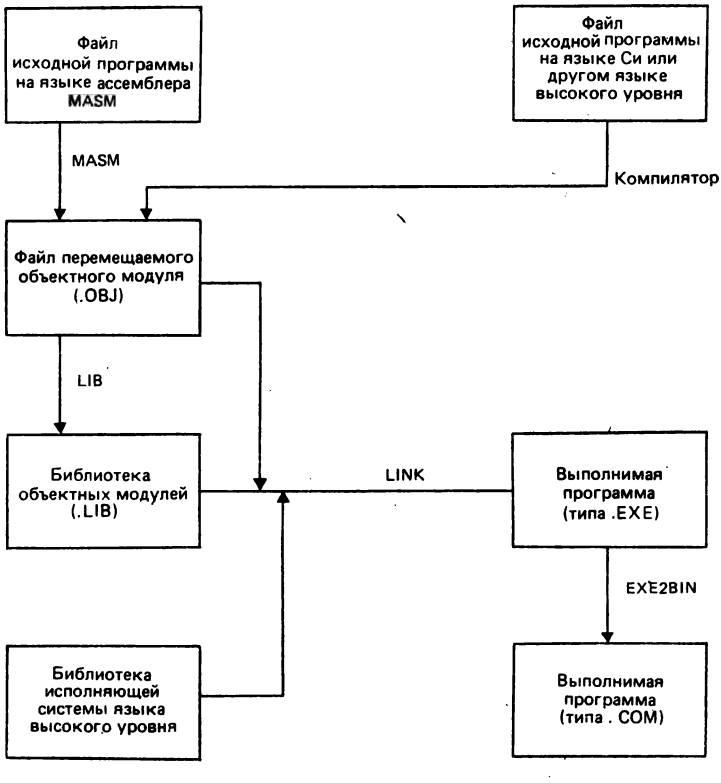


Рис. 4.8. Создание прикладной программы в среде MS-DOS: от исходного текста до выполнимого файла.

Предположим, что исходный текст программы HELLO.EXE из гл. 3 находится в файле HELLO.ASM. Для того чтобы с помощью ассемблера преобразовать исходную программу в перемещаемый объектный модуль HELLO.OBJ, содержащий символическую отладочную информацию, а также образовать в файле HELLO.LST листинг программы и в файле HELLO.CRF данные о перекрестных ссылках, нам необходимо ввести

```
C> MASM /C /L /Zi /T HELLO; <Enter>
```

Для преобразования файла перекрестных ссылок исходных данных HELLO.CRF в файл листинга перекрестных ссылок HELLO.REF следует ввести

```
C> CREF HELLO,HELLO <Enter>
```

Для преобразования перемещаемого объектного файла HELLO.OBJ в выполнимый файл HELLO.EXE, создания в файле HELLO.MAP карты загрузки и добавления в конец выполнимого файла символической отладочной информации следует ввести

```
C> LINK /MAP /CODEVIEW HELLO; <Enter>
```

Можно автоматизировать описанный процесс ввода, создав файл действия с именем HELLO (без расширения) и включив следующие команды:

```
hello.obj : hello.asm
 masm /C /L /Zi /T hello;
 cref hello,hello

hello.exe : hello.obj
 link /MAP /CODEVIEW hello;
```

После этого, если мы внесем изменения в файл HELLO.ASM и захотим вновь создать выполнимый файл HELLO.EXE, нам потребуется ввести только

```
C> MAKE HELLO <Enter>
```

ПОСОБИЯ ПО МАШИНАМ И ПРОГРАММИРОВАНИЮ

Литература по персональным компьютерам, совместимым с машинами IBM PC, семейству микропроцессоров Intel 80/86, языкам программирования ассемблера, Си весьма обширна. Далее приведена выборка книг, которые, на мой взгляд, являются наиболее полезными и надежными. Этот список не должен рассматриваться как утвержденный корпорацией Microsoft.

Пособия по обучению программированию на языке ассемблера MASM

Assembly Language Primer for the IBM PC and XT, by Robert Lafore.
New American Library, New York, NY, 1984. ISBN 0-452-25711-5.
8086/8088/80286 Assembly Language, by Leo Scanlon. Brady Books, Simon and Schuster, New York, NY, 1988. ISBN 0-13-246919-7.

Пособия по обучению программированию на языке Си

Microsoft C Programming for the IBM, by Robert Lafore. Noward K. Sams & Co., Indianapolis, IN, 1987. ISBN 0-672-22515-8.
Proficient C, by Augie Hansen.Microsoft Press, Redmond, WA, 1987.
ISBN 1-55615-007-5.

Справочник по микропроцессорам Intel 80/86

iAPX 88 Book. Intel Corporation, Literature Department SV3-3, 3065 Bowers Ave., Santa Clara, CA 95051. Order no.210200.
iAPX 286 Programmer's Reference Manual. Intel Corporation, Literature Department SV3-3, 3065 Bowers Ave., Santa Clara, CA 95051. Order no. 210498.
iAPX 386 Programmer's Reference Manual. Intel Corporation, Literature Department SV3-3, 3065 Bowers Ave., Santa Clara, CA 95051. Order no. 230985.

Архитектура машин PC, PC/AT и PS/2

The IBM Personal Computer From the Inside Out (Revised Edition), by Murray Sargent and Richard L. Shoemaker. Addison-Wesley Publishing Company, Reading, MA, 1986. ISBN 0-201-06918-0.
Programmer's Guide to PC & PS/2 Video Systems, by Richard Wilton.
Microsoft Press, Redmond, WA, 1987. ISBN 1-55615-103-9.
Personal Computer Technical Reference. IBM Corporation, IBM Technical Directory, P.O. Box 2009, Racine, WI 53404. Part no. 6322507.
Personal Computer AT Technical Reference. IBM Corporation, IBM Technical Directory, P.O. Box 2009, Racine, WI 53404. Part no. 6280070.
Options and Adapters Technical Reference. IBM Corporation, IBM Technical Directory, P.O. Box 2009, Racine, WI 53404. Part no. 6322509.
Personal System/2 Model 30 Technical Reference. IBM Corporation, IBM Technical Directory, P.O. Box 2009, Racine, WI 53404. Part no. 68X2201.
Personal System/2 Model 50/60 Technical Reference. IBM Corporation, IBM Technical Directory, P.O. Box 2009, Racine, WI 53404. Part no. 68X2224.
Personal System/2 Model 80 Technical Reference. IBM Corporation, IBM Technical Directory, P.O. Box 2009, Racine, WI 53404. Part no. 68X2256.

ГЛАВА 5. ВЫВОД С ПОМОЩЬЮ КЛАВИАТУРЫ И МАНИПУЛЯТОРА МЫШЬ

Для пользователя основным средством ввода в системе MS-DOS является клавиатура. Это естественно следует из интерфейса командной строки MS-DOS, родословную которой можно проследить вплоть до операционных систем мини-компьютеров с телетайпной консолью. В первые годы существования MS-DOS, когда обычным явлением были машины на базе микропроцессоров 8088/8086, почти все популярные прикладные программы использовали меню с управлением от клавиатуры и дисплеи в текстовом режиме.

Однако, по мере того как дешевели графические адаптеры с высоким разрешением (и машины на базе микропроцессоров 80286/80386 с достаточной мощностью для управления ими), популярность программ, поддерживающих окна и графический пользовательский интерфейс, неуклонно возрастала. Такие программы обычно пред-

полагают наличие какого-либо координатного устройства, например мыши, пера, джойстика или светового пера, и позволяют лавировать среди ваших задач в манере «прицельсяи стреляй», сводя ввод с клавиатуры к минимуму. В результате этого поддержка координатных устройств стала важным моментом для разработчиков программного обеспечения.

МЕТОДЫ ВВОДА С КЛАВИАТУРЫ

Прикладные программы, выполняющиеся в среде MS-DOS на компьютерах IBM PC и совместимых с ними машинах, могут использовать различные средства организации ввода с клавиатуры:

- Функции MS-DOS типа дескрипторов
- Традиционные символьно-ориентированные функции MS-DOS
- Функции драйвера клавиатуры ПЗУ BIOS

Указанные средства обеспечивают различную степень гибкости, переносимости и аппаратной независимости.

Функции типа дескрипторов, или потокоориентированные, идеологически восходящие к системам UNIX/XENIX, впервые введены в систему MS-DOS версии 2.0. Для использования этих функций программа должна предоставить дескриптор, или маркер, выбранного устройства, а также адрес и длину буфера.

Когда программа начинает выполняться, система MS-DOS предоставляет ей заранее определенные дескрипторы для наиболее часто используемых символьно-ориентированных устройств, включая клавиатуру.

Дескриптор	Имя устройства	Обозначение устройства
0	Стандартный ввод (stdin)	CON
1	Стандартный вывод (stdout)	CON
2	Стандартная ошибка (stderr)	CON
3	Стандартный последовательный порт (stdaux)	AUX
4	Стандартный принтер (stdprn)	PRN

Эти дескрипторы могут использоваться в операциях чтения и записи без всякой предварительной подготовки. Кроме того, программа может получить дескриптор символьно-ориентированного устройства, выполнив операцию открытия устройства на ввод или вывод с использованием его логического имени (как это делается при работе с файлом). Функции типа дескриптора поддерживают операцию перенаправления ввода-вывода, позволяя программе забирать входные данные вместо, например, клавиатуры с другого устройства или файла. Подробно вопросы перенаправления обсуждаются в гл. 15.

Традиционные функции символьного ввода – это надмножество функций символьного ввода-вывода, существовавших в системе CP/M. Первоначально включенные в систему MS-DOS для облегчения переноса прикладных программ из CP/M, они до сих пор широко используются. В MS-DOS версии 2.0 и старше большая часть традиционных функций также поддерживает операцию перенаправления ввода-вывода (хотя не так хорошо, как функции типа дескриптора).

Применение функций клавиатуры ПЗУ BIOS предполагает выполнение программы на машине, совместимой с IBM PC. Драйвер клавиатуры ПЗУ BIOS действует на значительно более примитивном уровне по сравнению с функциями MS-DOS и позволяет программе обойтись без операции перенаправления ввода-вывода или специальной обработки управляющих символов системой MS-DOS. Программы, использую-

щие драйвер клавиатуры ПЗУ BIOS, обладают принципиально худшей переносимостью, чем программы с функциями MS-DOS, и могут нарушать правильное выполнение других программ. Многие популярные, резидентные после завершения утилиты (terminate-and-stay-resident, TSR) попадают в эту категорию.

Ввод с клавиатуры с помощью дескрипторов

Основным средством MS-DOS ввода с клавиатуры с помощью дескрипторов является прерывание Int 21H с функцией 3FH (Чтение файла или устройства). В качестве параметров этой функции используются дескриптор, сегмент и относительный адрес буфера, а также длина буфера. (Более детальное описание этой функции см. в разд. 2 «Справочник по функциям MS-DOS».)

В качестве примера используем предопределенный дескриптор (0) стандартного ввода и системное прерывание Int 21H с функцией 3FH с целью прочитать с клавиатуры строку:

```
buffer  db   80 dup (?)     ; Буфер  ввода клавиатуры
        .
        .
        .
        mov  ah,3fh         ; Функция 3fh = чтение файла или устройства
        mov  bx,0           ; Дескриптор стандартного ввода
        mov  cx,80          ; Максимальное число байт при чтении
        mov  dx,seg buffer  ; DS:DX = адрес буфера
        mov  ds,dx
        mov  dx,offset buffer
        int  21h            ; Переход в MS-DOS
        jc   error          ; Переход, если ошибка обнаружена
        .
        .
        .
```

После возврата из прерывания Int 21H с функцией 3FH, если функция выполнена успешно, флаг переноса сбрасывается, а регистр AX содержит число прочитанных символов. Хотя при чтении с клавиатуры ошибок не должно быть, но если это случится, то установится флаг переноса, а в регистре AX будет код ошибки.

Стандартный ввод перенаправляем, и поэтому приведенная программа – не очень надежный способ ввода с клавиатуры. Ввод в программу в зависимости от параметра перенаправления, указанного пользователем в командной строке, может происходить с клавиатуры, из файла, с другого символьного устройства или даже из «мусорной корзины». Чтобы избежать операции перенаправления и абсолютно точно знать, откуда приходят входные данные, можно при вводе с клавиатуры отказаться от предопределенного дескриптора стандартного ввода и открыть консоль как файл, используя при этом дескриптор, полученный из операции открытия, как это показано в приведенной ниже программе:

```
buffer  db   80 dup (?)     ; Буфер ввода клавиатуры
fname   db   'CON',0        ; Логическое имя  клавиатуры
handle  dw   0              ; Дескриптор клавиатуры
        .
        .
        .
        mov  ah,3dh         ; Функция 3dh = открытие
        mov  al,0           ; Режим = чтение
```

```
mov  dx,seg fname   ; DS:DX = имя устройства
mov  ds,dx
mov  dx,offset fname
int  21h            ; Переход в MS-DOS
jc   error          ; Перейти, если операция открытия не выполнена
mov  handle,ax      ; Сохранить дескриптор  CON
.
.
.
mov  ah,3fh         ; Функция 3fh = чтение файла или устройства
mov  bx,handle      ; BX = Дескриптор  CON
mov  cx,80          ; Максимальное число байт при чтении
mov  dx,offset buffer ; DS:DX = адрес буфера
int  21h            ; Переход в MS-DOS
jc   error          ; Переход, если обнаружена ошибка
.
.
.
```

Когда программист применяет для чтения с клавиатуры прерывание Int 21H с функцией 3FH, конечный результат зависит от того, в каком состоянии находится дескриптор с точки зрения MS-DOS – в ASCII или двоичном (иногда называемом исходным). Режим ASCII является стандартным, хотя при необходимости с помощью прерывания Int 21H с функцией 44H (IOCTL) можно выбирать двоичный режим.

При работе в ASCII-режиме система MS-DOS помещает символы, полученные с клавиатуры, во внутренний буфер размером 128 байт, предоставляя пользователю возможность редактировать входные данные с помощью клавиши возврата (backspace), а также специальных функциональных клавиш. Система MS-DOS автоматически осуществляет отображение символов на стандартный ввод (эхо), расширяя при этом символы табуляции в требуемое число пробелов (хотя код ASCII табуляции 09H остается в буфере). Комбинации клавиш Ctrl-C, Ctrl-S и Ctrl-P получают специальную обработку, а клавиша Enter отображается в пару кодов «возврат каретки – перевод строки». При нажатии пользователем клавиши Enter или Ctrl-Z система MS-DOS осуществляет копирование запрошенных символов (или введенных символов, если их число меньше числа запрошенных) из внутреннего буфера в буфер вызывающей программы.

В двоичном режиме система MS-DOS не осуществляет ЭХО входных символов. MS-DOS передает в прикладную программу без изменений коды комбинаций клавиш Ctrl-C, Ctrl-S, Ctrl-P и Ctrl-Z, а также клавиш Enter, кроме того, прерывание Int 21H с функцией 3FH не возвращает управление прикладной програме до тех пор, пока не будут получены все затребованные символы.

Возможности обработки кода Ctrl-C обсуждаются в деталях в конце этой главы. Сейчас только отметим, что прикладной программист может заменить стандартный системный обработчик прерывания по коду Ctrl-C на собственный обработчик, что позволяет прикладной программе не потерять управления машиной при вводе пользователем Ctrl-C или Ctrl-Break.

Ввод с клавиатуры с помощью традиционных обращений

Традиционные системные функции клавиатуры предлагают разнообразный сервис работы с символами и символьными строками в режиме ЭХО и без него, а также обнаружение кода Ctrl-C. Эти функции представлены в следующем виде:

Прерывание Int 21H с функцией	Описание	Контроль кода Ctrl-C
01H	Ввод с клавиатуры с ЭХО	Да
06H	Непосредственный ввод-вывод через консоль	Нет
07H	Ввод с клавиатуры без ЭХО	Нет
08H	Ввод с клавиатуры без ЭХО	Да
0AH	Буферизированный ввод с клавиатуры	Да
0BH	Проверка состояния ввода	Да
0CH	Сброс буфера ввода и ввод	В зависимости от функции ввода

В системе MS-DOS версии 2.0 и старше операция перенаправления затрагивает все эти функции. Другими словами, эти функции работают словно специальный вид команды Int 21H с функцией 3FH, которая использует предопределенный дескриптор стандартного ввода (0).

Функции символьного ввода (01H, 06H, 07H и 08H) помещают код принятого символа в регистр AL. Например, в следующем фрагменте ожидается нажатие клавиши, и после того, как это произойдет, код символа этой клавиши помещается в регистр AL:

```
mov  ah,1          ; Функция 01h = чтение клавиатуры
int  21h           ; Переход в MS-DOS
```

Функции символьного ввода различаются по тому, как они выполняют эту операцию – с ЭХО или без него, а также реагируют ли они на прерывание по коду Ctrl-C или нет. Хотя в системе MS-DOS не предусмотрено специальной функции состояния клавиатуры, защищенной от прерывания по коду Ctrl-C, программа может прочитать состояние клавиатуры (в какой-то степени косвенным образом) без вмешательства со стороны операционной системы с помощью прерывания Int 21H с функцией 06H. Клавиши с расширенными кодами, такие, как специальные функциональные клавиши клавиатуры IBM PC, требуют двух обращений к функции символьного ввода.

В качестве альтернативы по символьному вводу программа может использовать буферизованный ввод строки (по прерыванию Int 21H с функцией 0AH), что позволяет за одну операцию ввести с клавиатуры целую строку. Система MS-DOS формирует такие строки во внутреннем буфере, причем передача их вызывающей программе произойдет только после нажатия пользователем клавиши Enter. При вводе символьной строки действуют все обычные клавиши редактирования, которые обрабатываются системным драйвером клавиатуры. Прерывание Int 21H с функцией 0AH можно применять следующим образом:

```
buff    db   81            ; Максимальная длина ввода
        db   0             ; Фактическая длина ( из MS-DOS )
        db   81 dup (0)    ; Для ввода с клавиатуры
        .
        .
        .
        mov  ah,0ah        ; Функция 0ah = чтение буферизованной строки
        mov  dx,seg buff   ; DS:DX = адрес буфера
```

```
mov  ds,dx
mov  dx,offset buff
int  21h            ; Переход в MS-DOS
.
.
.
```

Прерывание Int 21H с функцией 0AH имеет по сравнению с прерыванием Int 21H с функцией 3FH несколько важных отличительных особенностей. Во-первых, максимальная длина передается не в регистре CX, а в первом байте буфера. Во-вторых, общая длина помещается не в регистр AX, а во второй байт буфера. И наконец, в тех случаях, когда пользователь ввел на один символ меньше, чем это задано максимальным числом, MS-DOS игнорирует любые последующие символы и издает предупреждающий сигнал до тех пор, пока не будет нажата клавиша Enter.

Подробная информация о традиционных функциях ввода с клавиатуры – в разд. 2 «Справочник по функциям MS-DOS».

Ввод с клавиатуры с помощью функций ПЗУ BIOS

Программисты, пишущие прикладные программы для машин, совместимых с IBM PC, могут обойтись без функций клавиатуры системы MS-DOS, выбрав один из двух аппаратно-зависимых методов ввода с клавиатуры.

Первый метод связан с обращением к драйверу клавиатуры ПЗУ BIOS по прерыванию Int 16H. Например, следующий фрагмент считывает из буфера ввода клавиатуры один символ и помещает его в регистр AL:

```
mov  ah,0           ; Функция 0 = чтение клавиатуры
int  16h            ; Переход в ПЗУ BIOS
```

Прерывание Int 16H с функцией 00H также помещает в регистр AH скен-код клавиши клавиатуры, что позволяет программе обнаруживать коды клавиш, которые обычно системой MS-DOS не возвращаются. Другие функции прерывания Int 16H возвращают состояние клавиатуры (т. е. ожидает ли символ в буфере) или состояние специальных клавиш клавиатуры (из области данных ПЗУ BIOS по адресу 0000:0417H). Более детально функции клавиатуры ПЗУ BIOS рассматриваются в разд. 3 «Справочник по функциям ПЗУ BIOS и манипулятора мышь».

Вы должны тщательно все взвесить, прежде чем ввести в прикладную программу зависимость от ПЗУ BIOS. Хотя этот метод позволяет избежать любой операции перенаправления ввода-вывода, существуют способы достичь того же без введения зависимости от ПЗУ BIOS. К тому же обращение к драйверу клавиатуры ПЗУ BIOS сопряжено с реальными неудобствами:

- Полностью исключаются операции перенаправления ввода-вывода, что в отдельных случаях нежелательно.
- Используемая машина должна быть совместимой с IBM PC. На компьютерах ранних моделей, таких, как Hewlett-Packard Touch Screen или Wang Professional Computer, эти функции могут выполняться неправильно.
- Возникает сложное взаимодействие с утилитами типа TSR.

Другой метод ввода с клавиатуры IBM PC еще больше зависит от аппаратуры и предполагает написание нового обработчика прерываний Int 09H и прямое обслуживание прерываний контроллера клавиатуры. Это требует перевода скен-кодов в символы ASCII и обслуживания буфера клавиатуры. В обычных прикладных программах для PC нет резона переходить на такой уровень организации ввода с клавиатуры; поэтому здесь я не буду обсуждать этот метод подробнее. Если вы серьезно заинтересо-

ваны этой методикой, то лучшим справочником для вас будет листинг обработчика прерываний Int 09H ПЗУ BIOS из справочного руководства по IBM PC или PC/AT.

ОБРАБОТЧИКИ ПРЕРЫВАНИЙ ПО ВВОДУ КОМБИНАЦИЙ CTRL-C И CTRL-BREAK

При обсуждении способов ввода с клавиатуры при помощи функций традиционного обращения и типа дескриптора я несколько раз упомянул, что ввод комбинации Ctrl-C может помешать нормальному выполнению этих функций. Теперь давайте рассмотрим эту проблему подробнее.

Во время выполнения большинства операций символьного ввода-вывода система MS-DOS контролирует ввод комбинации Ctrl-C (код ASCII 03H), а при обнаружении этого кода выполняет прерывание Int 23H. Если системный флаг BREAK установлен, то MS-DOS следит за вводом Ctrl-C также и во время выполнения ряда других операций (как, например, чтение и запись файла). Обычно вектор прерывания Int 23H указывает на подпрограмму, которая просто завершает выполнение активного в данный момент процесса и возвращает управление родительскому процессу, которым, как правило, оказывается интерпретатор командной строки.

Другими словами, если во время выполнения своей программы вы случайно или намеренно введете комбинацию Ctrl-C, то MS-DOS снимет вашу программу с выполнения. Любой файл, открытый к этому моменту программой с помощью блоков управления (FCB), не будет закрыт должным образом, любой вектор прерывания, измененный в программе, может быть восстановлен неправильно, а, кроме того, если в программе выполняется какая-нибудь операция прямого ввода-вывода (например, если программа содержит драйвер прерывания последовательного порта), то могут случиться самые неожиданные вещи.

Хотя для отмены проверки ввода комбинации Ctrl-C вы можете применить несколько сравнительно эффективных методов, таких, как ввод с клавиатуры по прерыванию Int 21H с функциями 06H и 07H, задание двоичного режима для всех символьных устройств или сброс системного флага BREAK по прерыванию Int 21H с функцией 33H, ни один из них не обеспечивает абсолютную надежность. Самый простой и наиболее элегантный способ отмены контроля кода Ctrl-C – это подстановка собственного обработчика прерывания Int 23H, который задает действие, наиболее соответствующее вашей программе. По завершению выполнения программы MS-DOS автоматически восстанавливает предыдущее содержимое вектора прерывания Int 23H с помощью информации, сохраненной в префиксе сегмента программы. Следующий пример демонстрирует, каким образом устанавливается собственный обработчик прерывания по коду Ctrl-C (который в этом случае совсем ничего не делает):

```
push ds              ; Сохранить сегмент данных
                     ; Установить вектор прерывания int 23h ...
mov  ax,2523h        ; Функция 25h = установить прерывание
                     ; Int 23h = вектор обработчика прерывания
                     ; по вводу комбинации клавиш Ctrl-C
mov  dx,seg handler ; DS:DX = адрес обработчика прерываний
mov  ds,dx
mov  dx,offset handler
int  21h             ; Переход в MS-DOS

pop  ds              ; Восстановить сегмент данных
.
.
.
```

```
handler:                ; Обработчик прерываний по вводу
                        ; комбинации клавиш Crtl-C
        iret            ; Пустая операция
```

Первая часть программы, которая меняет содержимое вектора прерывания Int 23H, выполняется в прикладной программе в секции инициализации. Обработчик прерывания получает управление в тех случаях, когда система MS-DOS обнаруживает ввод с клавиатуры комбинации клавиш Ctrl-C. (Так как обработчик включает единственную команду возврата из прерывания, то код Ctrl-C остается во входном потоке символов и передается в прикладную программу в тот момент, когда она затребует символ с клавиатуры, при этом на экране появится ^ C.)

В тех случаях, когда вызывается обработчик прерывания Int 23H, MS-DOS находится в устойчивом состоянии. Таким образом, обработчик может вызывать любую функцию MS-DOS. Кроме того, он может перенастраивать сегментные регистры и указатель стека, а также передавать управление на другую точку входа в прикладную программу даже без возврата управления в MS-DOS с помощью IRET.

На машинах, совместимых с IBM PC, необходимо принимать в расчет наличие еще одного обработчика прерываний. Всегда, когда драйвер клавиатуры ПЗУ BIOS обнаруживает комбинацию клавиш Ctrl-Break, он вызывает обработчик, адрес которого хранится в векторе прерывания Int 1BH. Стандартный обработчик прерывания Int 1BH ПЗУ BIOS ничего не делает. Система MS-DOS изменяет содержимое вектора прерывания Int 1BH так, чтобы он указывал на свой собственный обработчик прерывания, который устанавливает флаг и возвращает управление; конечный эффект заключается в отображении кода Ctrl-Break в код Ctrl-C, который к тому же обрабатывается раньше других символов, находящихся в буфере клавиатуры.

Перехват прикладной программой вектора прерывания Int 1BH является несколько рискованным, но исключительно полезным делом. Так как клавиатура управляется по прерываниям, то нажатие клавиш Ctrl-Break позволяет прикладной программе вернуть управление практически при любых обстоятельствах, часто даже если программа завершилась аварийно или зациклилась.

Как правило, не удается использовать одну и ту же программу для обработки прерываний Int 1BH и Int 23H. Обработчик прерывания Int 1BH более ограничен в своих возможностях, так как обращение к нему является результатом аппаратного прерывания, а система MS-DOS в момент обслуживания такого прерывания может выполнять критическую секцию [1] программы. В этом случае все регистры, за исключением CS:IP, находятся в неизвестном состоянии; по-видимому, их необходимо сохранить, а затем модифицировать, до того как ваш обработчик прерывания сможет выполняться. Точно также неизвестна глубина используемого стека при обращении к обработчику прерывания Int 1BH, и поэтому если обработчик должен осуществлять операции, интенсивно использующие стек, то, возможно, необходимо сохранить сегмент и указатель стека, переключив затем указатель на новый стек, имеющий достаточную глубину.

В обычных прикладных программах, вероятно, не следует оставаться в обработчике прерывания Int 1BH вместо выполнения IRET. Вследствие тонких различий между ПЗУ BIOS, выпускаемых различными фирмами, трудно предсказать состояние контроллера клавиатуры и программируемого контроллера прерываний (PTC) 8259 в момент начала работы обработчика прерывания Int 1BH. Кроме того, сама система MS-DOS может находиться в неустойчивом состоянии в момент прерывания, что может

[1] Т. е. часть программы, которая должна выполняться без вмешательства со стороны других процессов, в том числе, обслуживания прерываний. – *Прим. перев.*

проявиться в виде неожиданных критических ошибок во время последующих операций ввода-вывода. И наконец, MS-DOS версии 3.2 и старше выделяет область стека для нужд обработчика прерывания Int 09H за счет внутреннего пула [1]. Если из обработчика прерывания Int 1BH так и не произойдет выхода, то не произойдет выхода и из обработчика прерывания Int 09H, и повторные команды Ctrl-Break в конечном счете исчерпают стек, остановив систему.

Так как прерывание Int 1BH обрабатывается не MS-DOS, а ПЗУ BIOS, то система MS-DOS после завершения программы не восстанавливает предыдущее содержимое вектора прерывания Int 1BH. Если ваша программа изменяет содержимое этого вектора, то она должна сохранить исходное значение вектора и восстановить его до своего завершения. В противном случае вектор будет продолжать указывать на некоторую случайную область в следующей программе при ее выполнении, и следующее нажатие пользователем клавиш Ctrl-Break в лучшем случае приведет к аварии системы.

Обработчики прерываний по вводу комбинаций Ctrl-C и Ctrl-Break и языки высокого уровня

В тех случаях, когда вы программируете на языке ассемблера, перехват прерываний по вводу комбинаций Ctrl-C и Ctrl-Break не представляет труда. Эта операция лишь немногим сложнее в случае языков высокого уровня, если только у вас имеется достаточно информации о межъязыковом интерфейсе, чтобы включить в состав программы небольшой фрагмент на языке ассемблера.

Листинг BREAK.ASM на рис. 5.1 содержит текст обработки прерывания по Ctrl-Break, который можно скомпоновать с программой малой модели на языке Си фирмы

```
        page    55,132
        title--Обработчики прерываний Ctrl-C и Ctrl-Break

        name    break


;
; Обработчики прерываний по вводу комбинации клавиш
; Ctrl-C и Ctrl-Break для программ на языке Си фирмы
; Microsoft, предназначенных для выполнения на компью-
; терах IBM PC и совместимых с ними машинах.
;
; автор: Рей Данкан
;
; Команда на трансляцию:   C>MASM /Mx BREAK;
;
; Этот модуль позволяет программам на языке Си перехваты-
; вать управление при вводе пользователем комбинации клавиш
; Ctrl-Break или Ctrl-C; используются соглашения языка Си
; фирмы Microsoft о передаче параметров. Предполагается,
; что Си - программа соответствует малой модели памяти.
```

Рис. 5.1. BREAK.ASM. Обработчик прерываний Ctrl-C и Ctrl-Break, который можно компоновать с программами на языке Си фирмы Microsoft.

[1] Пул – динамическая область, используемая системой для собственных целей, которую иногда еще называют «куча». – *Прим. перев.*

```
;
; Процедура _capture вызывается для установки нового обра-
; ботчика прерываний при вводе пользователем комбинации
; клавиш Ctrl-C или Ctrl-Break (коды ASCII 1bh и 23h).
; Процедуре _capture передается адрес статической перемен-
; ной, значение которой устанавливается равным "истина"
; при обнаружении обработчиком прерываний ввода комбинации
; клавиш Ctrl-C или Ctrl-Break.
; Синтаксис Си - программы:
;             static int flag;
;             capture(&flag);
; Процедура _release вызывается Си -программой для восста-
; новления исходного состояния обработчика прерываний
; Ctrl-Break и Ctrl-C. Синтаксис Си - программы:
;
;             release();
;
; Процедура ctrlbrk, фактически обработчик прерывания, по-
; лучает управление в тех случаях, когда ПЗУ BIOS осущест-
; вляет прерывание int 1bh или система MS-DOS осуществляет
; прерывание int 23h. Процедура всего лишь устанавливает
; значение переменной true(1) из Си -программы и возвращает
; управление.
;

args    equ     4               ; Относительный адрес стека аргументов
                                ; Си - программы,соответствующей малой
                                ; модели памяти
cr      equ     0dh             ; Код ASCII возврата каретки
lf      equ     0ah             ; Код ASCII перевода строки

_TEXT   segment word public 'CODE'

        assume cs:_TEXT


        public  _capture
_capture proc   near            ; Захват векторов прерываний Ctrl-Break
                                ; и Ctrl-C

        push    bp              ; Установить кадр стека
        mov     bp,sp

        push    ds              ; Сохранение регистров
        push    di
        push    si

                                ; Сохранение адреса флага "flag"
                                ; вызывающей программы
        mov     ax,word ptr [bp+args]
        mov     word ptr cs:flag,ax
        mov     word ptr cs:flag+2,ds

                                ; Сохранение адреса исходного
        mov     ax,3523h        ; обработчика прерываний int 23h
        int     21h
        mov     word ptr cs:int23,bx
        mov     word ptr cs:int23+2,es
```

Рис. 5.1.

```
        mov     ax,351bh        ; Сохранить адрес исходного
        int     21h             ; обработчика прерываний int 1bh
        mov     word ptr cs:int1b,bx
        mov     word ptr cs:int1b+2,es
        push    cs              ; Установить DS:DX = адрес
        pop     ds              ; нового обработчика прерываний
        mov     dx,offset _TEXT:ctrlbrk

        mov     ax,02523h       ; Установить вектор прерывания int 23h
        int     21h

        mov     ax,0251bh        ; Установить вектор прерывания int 1bh
        int     21h

        pop     si              ; Восстановление регистров
        pop     di
        pop     ds

        pop     bp              ; Отбросить кадр стека
        ret                     ; и вернуться к месту вызова

_capture endp

        public _release
_release proc  near             ; Восстановление исходного состояния
                                ; обработчиков прерываний Ctrl-C и
                                ; Ctrl-Break

        push    bp              ; Сохранение регистров
        push    ds
        push    di
        push    si

        lds     dx,cs:int1b     ; Получить адрес исходного
                                ; обработчика прерывания  int 1bh

        mov     ax,251bh        ; Установить вектор прерывания int 1bh
        int     21h

        lds     dx,cs:int23     ; Получить адрес исходного
                                ; обработчика прерывания int 23h

        mov     ax,2523h        ; Установить вектор прерывания int 23h
        int     21h

        pop     si              ; Восстановить регистры
        pop     di              ; и вернуться к месту вызова
        pop     ds
        pop     bp
        ret

_release endp

ctrlbrk proc    far             ; Обработчик прерываний
                                ; Ctrl-C и Ctrl-Break

        push    bx              ; Сохранить регистры
        push    ds
```

Рис. 5.1.

```
        lds     bx,cs:flag      ; Получить адрес "флаговой переменной"
                                ; Си - программы

                                ; и установить значение флага "истина"
        mov     word ptr ds:[bx],1

        pop     ds              ; Восстановить регистры
        pop     bx

        iret                    ; Возврат из обработчика прерываний

ctrlbrk endp

flag    dd      0               ; Дальний указатель на флаг состояния
                                ; Ctrl-C или Ctrl-Break  вызывающей
                                ; программы
int23   dd      0               ; Адрес исходного обработчика
                                ; прерываний Ctrl-C

int1b   dd      0               ; Адрес исходного обработчика
                                ; прерываний Ctrl-Break

_TEXT   ends

        end
```

Рис. 5.1.

```
/*
    TRYBREAK.C

    Демонстрационная программа BREAK.ASM  - обработчик
    прерываний Ctrl-Break и Ctrl-C, автор Рей Данкан

    Вызов на выполнение файла TRYBREAK.EXE:

    MASM /Mx BREAK;
    CL TRYBREAK.C BREAK.OBJ
*/

#include <stdio.h>

main(int argc, char *argv[])
{
    int hit = 0;                /* Флаг нахатия клавиши         */
    int c = 0;                  /* Символ с клавиатуры          */
    static int flag = 0;        /* "Истина", если обнарухен ввод
                                   комбинации клавиш
                                   Ctrl-Break или Ctrl-C        */

    puts("\n*** TRYBREAK.C запущена ***\n");
    puts("Ввести Ctrl-C или Ctrl-Break для проверки обработчика,");
    puts("Нахать клавишу Esc для завершения TRYBREAK.\n");
```

Рис. 5.2. TRYBREAK.C. Простая программа на языке Си фирмы Microsoft, которая демонстрирует применение обработчика прерываний BREAK.ASM, приведенного на рис. 5.1.

```
    capture(&flag);             /* Установить новый обработчик
                                   прерываний Ctrl-C и
                                   Ctrl-Break и передать
                                   адрес флага                */

    puts("TRYBREAK захватила векторы прерываний.\n");

    while(1)
    {
        hit = kbhit();          /* Проверить нажатие клавиши  */
                                /* (MS-DOS обнаруживает Ctrl-C
                                   при опросе клавиатуры)     */

        if(flag != 0)           /* Если флаг = "истина", то   */
        {                       /* произошло прерывание       */
            puts("\nОбнаружен Control-Break.\n");
            flag = 0;           /* Сбросить флаг прерывания   */

        }
        if(hit != 0)            /* Если любой символ ожидает
                                   (в буфере клавиатуры)      */
        {
            c = getch();        /* Прочитать код клавиши,завершить,
                                   если это код клавиши ESC   */
            if( (c & 0x7f) == 0x1b) break;
            putch(c);           /* Иначе вывести на экран      */
        }
    }
    release();                  /* Восстановить исходный
                                   обработчик прерываний
                                   Ctrl-C и Ctrl-Break        */

    puts("\n\nTRYBREAK векторы прерываний освободила.");
}
```

Рис. 5.2.

Microsoft, для машин, совместимых с IBM PC. Короткая Си-программа, приведенная на рис. 5.2, демонстрирует применение обработчика прерываний. (Эта программа должна легко переноситься на другие компиляторы с языка Си.)

При вызове процедуры capture, включенной в состав обработчика, ей передается адрес целочисленной переменной из программы на языке Си. Процедура сохраняет адрес переменной, заносит в векторы прерываний Int 1BH и Int 23H адрес собственного обработчика прерываний и затем возвращает управление.

В тех случаях, когда MS-DOS обнаруживает коды Ctrl-C или Ctrl-Break, обработчик прерываний присваивает целочисленной переменной Си-программы значение «истина» (1) и возвращает управление. Далее Си-программа может спокойно опрашивать эту переменную. Разумеется, для неоднократного обнаружения кода Ctrl-C программа должна сбрасывать переменную в ноль.

Процедура release восстанавливает исходные значения векторов прерываний Int 1BH и Int 23H, блокируя тем самым обработчик прерываний. Хотя, строго говоря, процедуре release нет необходимости что-то делать с вектором Int 23H, это действие дает возможность Си-программе восстановить стандартный обработчик прерывания Int 23H еще до своего завершения.

КООРДИНАТНЫЕ УСТРОЙСТВА

Драйверы для координатных устройств поставляются производителями аппаратных средств и загружаются с помощью предложения DEVICE в файле CONFIG.SYS. Хотя аппаратные характеристики существующих координатных устройств сильно различаются, драйверы почти всех этих устройств пользуются одинаковым программным интерфейсом с прикладными программами – протоколом прерывания Int 33H, используемым драйвером манипулятора мышь фирмы Microsoft. Драйвер манипулятора мышь фирмы Microsoft версии 6, используемый в то время, когда писалась эта книга, содержит следующие функции:

Функция	Описание
00H	Привести мышь в исходное состояние и получить состояние драйвера
01H	Вывести на экран указатель мыши
02H	Погасить указатель мыши
03H	Получить положение указателя мыши и состояние клавиш
04H	Установить положение указателя мыши
05H	Получить информацию о нажатиях клавиш мыши
06H	Получить информацию об отпусканиях клавиш мыши
07H	Установить горизонтальные границы для указателя
08H	Установить вертикальные границы для указателя
09H	Установить форму графического указателя
0AH	Установить тип текстового указателя
0BH	Прочитать состояние счетчиков движения мыши
0CH	Установить определяемый пользователем обработчик событий мыши
0DH	Включить эмуляцию светового пера
0EH	Выключить эмуляцию светового пера
0FH	Установить отношение микки на пиксел
10H	Установить область подавления указателя мыши
13H	Установить порог удвоения скорости
14H	Сменить определяемые пользователем обработчики событий мыши
15H	Получить размер буфера хранения состояния мыши
16H	Сохранить состояние драйвера мыши
17H	Восстановить состояние драйвера мыши
18H	Установить дополнительный обработчик событий мыши
19H	Получить адрес дополнительного обработчика событий мыши
1AH	Установить чувствительность мыши
1BH	Получить чувствительность мыши
1CH	Установить частоту прерываний от мыши
1DH	Выбрать видеостраницу для указателя
1EH	Получить видеостраницу для указателя
1FH	Заблокировать драйвер мыши
20H	Разблокировать драйвер мыши
21H	Привести в исходное состояние драйвер мыши
22H	Установить язык сообщений драйвера мыши
23H	Получить номер языка
24H	Получить версию драйвера, тип мыши и номер IRQ

Хотя этот список функций манипулятора мышь может показаться пугающе большим, обычной прикладной программе понадобится всего лишь несколько из них.

Прежде всего программа обращается к прерыванию Int 33H с функцией 00H

с целью проинициализировать драйвер мыши в соответствии с видеорежимом и проверить состояние драйвера. На этой стадии мышь оживает и программа может получать данные о состоянии мыши и ее положении. Однако указатель мыши не будет видимым до обращения к прерыванию Int 33H с функцией 01H.

Затем для наблюдения за положением мыши и состоянием клавиш мыши программа может обратиться к прерыванию Int 33H с функциями 03H, 05H и 06H. С другой стороны, программа может установить обработчик прерываний событий мыши, используя прерывание Int 33H с функцией 0CH. Этот способ исключает необходимость опроса драйвера манипулятора мышь; драйвер будет сообщать программе, вызывая обработчик прерываний, о любом перемещении мыши, а также нажатии или отпускании ее клавиш.

В тех случаях, когда программа прекращает работу с манипулятором мышь, она может вызвать прерывание Int 33H с функцией 02H для того, чтобы погасить указатель мыши. Если программа установила обработчик прерываний манипулятора мышь, то следует заблокировать последующие вызовы обработчика, сбросив драйвер мыши еще раз с помощью прерывания Int 33H с функцией 00H.

Полное описание функций драйвера мыши см. в разд. 3 «Справочник по функциям ПЗУ BIOS и манипулятора мышь». На рис. 5.3 приведена небольшая демонстрационная программа, в которой для вывода на экран положения и состояния мыши происходит непрерывный ее опрос.

```
/*
    Демонстрационная программа (Си) прерывания Int 33H
    драйвера манипулятора мышь. Автор: Рей Данкан.1988 г.

    Компилируется с модулем: CL MOUDEMO.C
*/

#include <stdio.h>
#include <dos.h>

union REGS regs;

void cls(void);                    /* Прототипы функций
void gotoxy(int, int);

main(int argc, char *argv[])
{
    int x,y,buttons;               /* Некоторые рабочие переменные      */
                                   /* для состояния мыши                */

    regs.x.ax = 0;                 /* Сброс драйвера мыши               */
    int86(0x33, &regs, &regs);     /* и проверка состояния              */

    if(regs.x.ax == 0)             /* Завершить, если мыши нет        */
    {   printf("nMouse not available\n");
        exit(1);
    }
    cls();                         /* Очистить экран                    */
```

Рис. 5.3. MOUDEMO.C. Простая программа на языке Си фирмой Microsoft, которая опрашивает манипулятор мышь и непрерывно отображает координаты указателя мыши в верхнем левом углу экрана.

Программа использует видеодрайвер ПЗУ BIOS, рассматриваемый в гл. 6, для очистки экрана и позиционирования курсора в текстовом режиме.

```
    gotoxy(45,0);                   /* и вывести подсказку                */
    puts("Для завершения нахать одновременно обе клавиши мыши");

    regs.x.ax = 1;                  /* Показать на экране курсор мыши     */
    int86(0x33, &regs, &regs);

    do {
       regs.x.ax = 3;               /* Получить полохение мыши            */
       int86(0x33, &regs, &regs); /* и состояние ее клавиш                */
       buttons = reg.x.bx & 3;
       x = regs.x.cs;
       y = regs.x.dx;
       gotoxy(0,0);                 /* Вывести на экран полохение мыши    */
       printf("X = %3d  Y =%3d",x,y);

       } while(buttons !=3);        /* Завершить,если обе клавиши нахаты */

       regs.x.ax = 2;               /* Скрыть курсор мыши                 */
       int86(0x33, &regs, &regs);

       cls();                       /* Выдать сообщение на экран
                                       и завершить                        */
       gotoxy(0,0);
       puts("Have a Mice Day!);
}

/*
       Очистка экрана
*/
void cls(void)
{
    regs.x.ax = 0x0600;             /* Видеодрайвер ПЗУ BIOS              */
    regs.h.bh = 7 ;                 /* Прерывание int 10h с функцией 06h */
    regs.x.cx = 0 ;                 /* Инициализировать окно              */
    regs.h.dh = 24;
    regs.h.dl = 79;
    int86(0x10, &regs, &regs);
}

/*
   Полохение курсора (x,y)
*/
void gotoxy(int x,int y)
{
    regs.h.dl = x;                  /* Видеодрайвер ПЗУ BIOS              */
    regs.h.dh = y;                  /* Прерывание int 10h с функцией 02h */
    regs.h.bh = 0;                  /* Координаты курсора                 */
    regs.h.ah = 2;
    int86(0x10, &regs, &regs);
}
```

Рис. 5.3.

ГЛАВА 6. ВИДЕОСИСТЕМА

Важным элементом прикладной программы является видеопредставление. Часто мнение пользователя о производительности и «изяществе» программы основано на скорости и привлекательности образуемых ею изображений. Кроме того, для программиста важно чувствовать средства и возможности визуализации вычислительной системы на всех уровнях от MS-DOS до голой аппаратуры.

ВИДЕОАДАПТЕРЫ

Адаптеры монитора, встречающиеся в компьютерах IBM PC и совместимых с ними машинах, имеют гибридный интерфейс с центральным процессором. Общие характеристики изображения, такие, как разрешение по вертикали и горизонтали, цвет фона и палитра, определяются значениями, записанными в порты ввода-вывода, адреса которых зашиты в адаптере, в то время как видеоизображение отдельного символа или графической точки (пиксела) управляется конкретной ячейкой области памяти, называемой буфером регенерации или буфером изображения. И центральный процессор, и видеоконтроллер имеют доступ к этой области; программа обновляет изображение на экране дисплея, записывая непосредственно в буфер изображения символьные или двоичные коды. (Эта операция называется вводом-выводом с отображением на общее с памятью адресное пространство или просто с отображением на память.)

В момент написания книги повсеместно использовались следующие адаптеры:

- Адаптер монохромного дисплея (Monochrome/Printer Display Adapter, MDA). Установленный в 1981 г. на исходных компьютерах фирмы IBM, адаптер поддерживает монохромный дисплей, работающий в текстовом режиме с зеленым экраном 80 × 25 и не имеет никаких средств графики.
- Цветной графический адаптер (Color/Graphics Adapter, CGA). Представленный фирмой IBM также в 1981 г., адаптер поддерживает текстовый режим 40 × 25 и 80 × 25, а также графический режим 320 × 200, 4 цвета или 640 × 200, 2 цвета (с адресацией ко всем точкам) на композитных и цифровых RGB-мониторах [1].
- Улучшенный графический адаптер (Enchanced Graphics Adapter, EGA). Представленный в 1985 г. фирмой IBM этот адаптер, совместимый «снизу вверх» с CGA [2], добавляет поддержку графического режима 640 × 350, 16 цветов на цифровых RGB-мониторах. Кроме того, он поддерживает текстовый режим, совместимый с MDA.
- Многоцветная графическая матрица (Multi-Color Graphics Array, MCGA). Установленный в 1987 г. фирмой IBM на компьютерах Personal System/2 (PS/2) моделей 25 и 30 этот адаптер частично совместим с CGA и EGA и поддерживает графический режим 640 × 480, 2 цвета или 320 × 200, 256 цветов на аналоговых RGB-мониторах.

Видеографическая матрица (Video Graphics Array, VGA). Установленный в 1987 г. фирмой IBM на компьютерах PS/2 моделей 50, 60 и 80 этот адаптер совместим «снизу вверх» с EGA и поддерживает графический режим 640 × 480, 16 цветов или 320 × 200, 256 цветов на аналоговых RGB-мониторах. Кроме того, он поддерживает текстовый режим, совместимый с MDA.

- Графические платы Hercules Graphics Card, Graphics Card Plus и InColor Cards.

---

[1] Цветной графический адаптер имеет выходы для двух видов подключения: в одном случае формируется единый композитный сигнал цветного изображения, в другом – отдельные сигналы трех основных цветов – красный, зеленый, синий, которые называются RGB-сигналами. Подключаемые к такому выходу мониторы называются RGB-мониторами. – *Прим. перев.*

[2] Т. е. всякая прикладная программа, предназначенная для работы с адаптером CGA, будет также работать и с адаптером EGA, но наоборот – неверно. – *Прим. перев.*

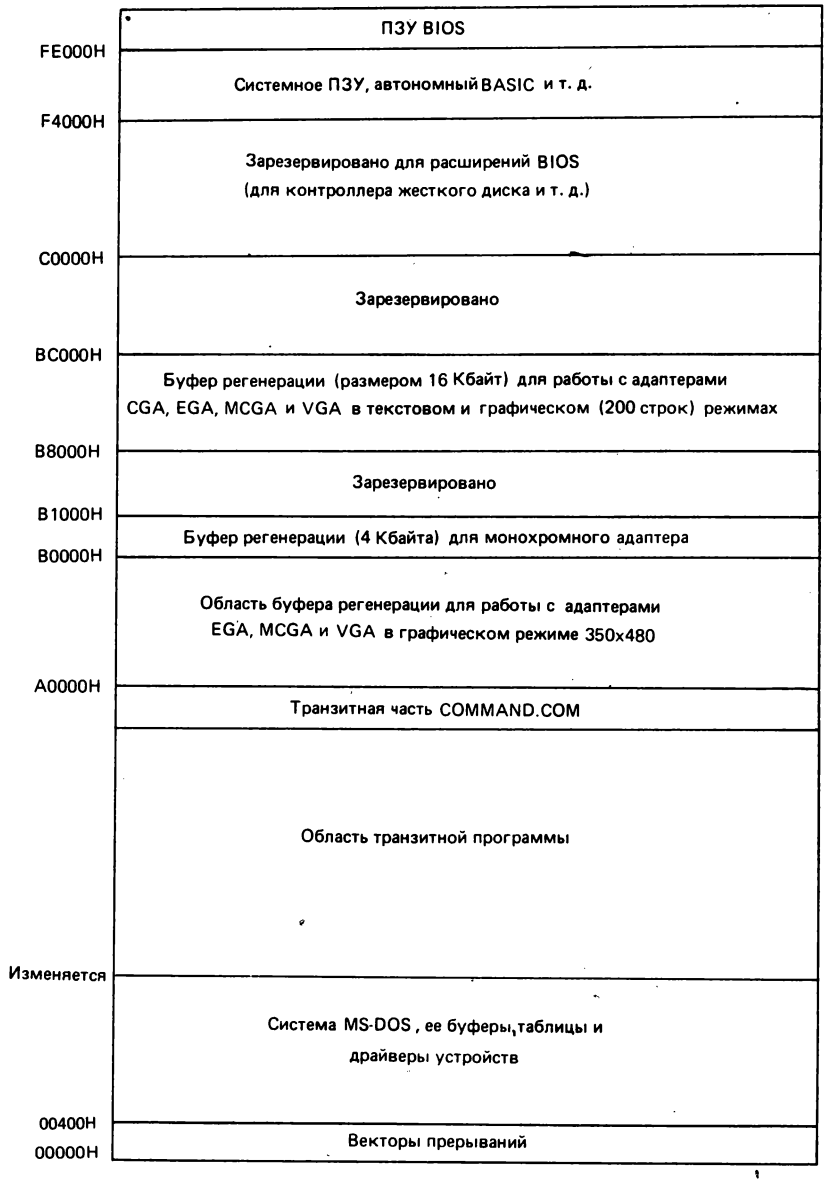


Рис. 6.1. Диаграмма памяти IBM PC и совместимых персональных компьютеров, демонстрирующая расположение буферов регенерации для различных адаптеров.

Эти адаптеры совместимы «снизу вверх» с MDA в текстовом режиме, однако предлагают графические средства, несовместимые с другими адаптерами фирмы IBM.

Расположение буферов регенерации для различных адаптеров, совместимых с IBM PC, показано на рис. 6.1.

ВОПРОСЫ ПОДДЕРЖКИ

Система MS-DOS предоставляет несколько функций передачи текста на дисплей. Версия 1 поддерживала вывод только телетайпного типа; в версии 2 добавилась поддержка необязательного драйвера консоли ANSI, что позволило программисту выполнять операции очистки экрана, позиционирования курсора, выбора цвета и атрибутов с помощью стандартных управляющих Esc-последовательностей, включаемых в выводимую строку. Программы, которые используют только функции MS-DOS, будут правильно работать на любой машине с этой системой независимо от уровня совместимости с аппаратурой фирмы IBM.

ПЗУ BIOS машин, совместимых с IBM PC, содержит видеодрайвер, который программы могут возбуждать непосредственно, в обход средств MS-DOS. Функции ПЗУ BIOS позволяют программе записывать текст или отдельные пикселы на экран, а также выбирать видеорежимы, видеостраницы, палитру, цвета́ изображения символа и фона. Эти функции довольно эффективны (по крайней мере по сравнению с функциями MS-DOS), хотя их возможности поддерживать графику весьма слабые.

К сожалению, графические функции как MS-DOS, так и ПЗУ BIOS выполнены в расчете на модель терминала с адресуемым курсором, и поэтому они не используют полностью возможностей операции с отображением на память адаптеров широкополосных дисплеев, применяемых на машинах, совместимых с IBM PC. В результате почти все популярные диалоговые программы с развитой графикой игнорируют и MS-DOS, и ПЗУ BIOS, а обращаются непосредственно к регистрам видеоконтроллера и буферу изображения.

Программы, которые управляют аппаратурой непосредственно, иногда называют «некорректными», так как они осуществляют операции, обычно возлагаемые на системные драйверы устройств. Такие задачи вступают в серьезное противоречие с диспетчерами многозадачных сред реального режима, таких, как DesqView и Microsoft Windows, и являются основой того, что эти системы не нашли широкого применения. Однако можно спорить с тем, что вина за некорректность лежит только на прикладных программах и не связана с недостатками MS-DOS и ПЗУ BIOS, которые даже по прошествии шести лет с момента появления компьютеров IBM PC не предоставляют видеофункций требуемых диапазона и мощности.

ВИДЕОФУНКЦИИ MS-DOS

В системе MS-DOS версии 2.0 и старше предпочтительным является метод пересылки текста на дисплей с помощью дескриптора по прерыванию Int 21H с функцией 40H (запись файла или внешнего устройства). К моменту передачи управления прикладной программе MS-DOS уже назначила дескрипторы стандартного вывода (1) и стандартной ошибки (2), которые могут использоваться немедленно. Например, фрагмент программы, приведенный ниже, выводит на экран сообщение hello с помощью дескриптора стандартного вывода.

```
msg     db      'hello'      ; Вывести сообщение на дисплей
msg_len equ     $-msg        ; Длина сообщения
        .
        .
        .
        mov     ah,40        ; Функция 40h = запись файла или устройства
        mov     bx,1         ; BX = дескриптор стандартного вывода
        mov     cx,msg_len   ; CX = длина сообщения
        mov     dx,seg msg   ; DS:DX = адрес сообщения
```

```
        mov     ds,dx
        mov     dx,offset msg
        int     21h             ; Переход в MS-DOS
        jc      error           ; Переход, если обнаружена ошибка
        .
        .
        .
```

Если эта операция выполняется без ошибки, то функция сбрасывает флаг переноса и помещает в регистр AX значение, равное числу переданных символов. Если в тексте не встречается комбинации Ctrl-Z, а также если нет операции перенаправления стандартного вывода на дисковый файл и диск полностью не заполнен, это число должно соответствовать числу затребованных символов.

Как и в случае ввода с клавиатуры, возможность задать в командной строке параметры перенаправления, невидимые прикладной программе, означает, что если вы используете предопределенный дескриптор стандартного вывода, то вы никогда точно не знаете, куда направлен ваш вывод. Однако для того, чтобы гарантировать вывод именно на экран, можно применять предопределенный дескриптор ошибки, который всегда открыт на устройство CON (логическая консоль) и неперенаправляем.

В качестве альтернативы дескрипторам стандартного вывода и стандартной ошибки можно обойти любое перенаправление вывода и открыть для CON отдельный канал, используя дескриптор, полученный от операции открытия символьного вывода. Например, следующая программа открывает консольный дисплей на вывод, а затем записывает туда строку hello:

```
fname   db      'CON',0         ; Имя устройства CON
handle  dw      0               ; Дескриптор устройства CON
msg     db      'hello'         ; Вывести сообщение на дисплей
msg_len equ     $-msg           ; Длина сообщения
        .
        .
        .
mov     ax,3d02h                ; AH = функция 3dh = открыть
                                ; AL = режим = чтение - запись
mov     dx,seg fname            ; DS:DX = имя устройства
mov     ds,dx
mov     dx,offset fname
int     21h                     ; Переход в MS-DOS
jc      error                   ; Перейти, если операция открыть невыполнена
mov     handle,ax               ; Сохранить дескриптор CON
        .
        .
        .
mov     ah,40h                  ; Функция 40h = запись
mov     cx,msg_len              ; CX = длина сообщения
mov     dx,seg msg              ; DS:DX = адрес сообщения
mov     ds,dx
mov     dx,offset msg
mov     bx,handle               ; BX = дескриптор устройства CON
int     21h                     ; Переход в MS-DOS
jc      error                   ; Перейти, если обнаружена ошибка
        .
        .
```

Как и в случае функций ввода с клавиатуры, система MS-DOS поддерживает

традиционные видеофункции, совместимые «снизу вверх» с соответствующими функциями CP/M:

- Прерывание Int 21H с функцией 02H пересылает символ в регистре DL на устройство стандартного вывода. Функция чувствительна к прерываниям по комбинации Ctrl-C и обрабатывает соответствующим образом символы возврата каретки, переноса строки, звонка и возврата на один символ.
- Прерывание Int 21H с функцией 06H передает символ в регистре DL на устройство стандартного вывода, но оно нечувствительно к комбинации Ctrl-C. При работе с этой функцией надо быть предельно внимательным, так как ее можно применять как для ввода, так и для запросов состояния.
- Прерывание Int 21H с функцией 09H пересылает на устройство стандартного вывода строку символов, которая завершается символом $.

В системе MS-DOS версии 2 и старше эти три функции традиционного обращения внутренне преобразуются в операции записи на стандартный вывод и поэтому чувствительны к операции перенаправления вывода.

В следующем фрагменте программы выдается предупреждающий звуковой сигнал путем посылки ASCII-кода звонка (07H) на видеодрайвер с помощью традиционного символьного вывода по прерыванию Int 21H с функцией 02H.

```
.
.
.
mov     dl,7                ; 07h = код ASCII звонка
mov     ah,2                ; Функция 02h = вывод символа на экран
int     21h                 ; Переход в MS-DOS
.
.
.
```

В следующем фрагменте для вывода на экран строки символов используется традиционная функция вывода строки (прерывание Int 21H с функцией 09H):

```
msg     db      'hello$'
        .
        .
        .
        mov     dx,seg msg  ; DS:DX = Адрес сообщения
        mov     ds,dx
        mov     dx,offset msg
        mov     ah,9        ; Функция 09h = записать строку
        int     21h         ; Переход в MS-DOS
        .
        .
        .
```

Заметим, что система MS-DOS воспринимает символ $ как признак завершения и не выводит его на экран.

Управление экраном с помощью функций MS-DOS

Если система MS-DOS версии 2.0 и старше в ответ на запрос директивы DEVICE в файле CONFIG.SYS загружает необязательный драйвер устройства ANSI.SYS, то программы получают возможность очищать экран, управлять положением курсора, а также выбирать цвета изображения символа и фона, встраивая управляющие

Esc-последовательности в выводимые строки текста. Esc-последовательности называются так потому, что они начинаются с символа Escape (1BH), который предупреждает драйвер о необходимости перехватить и интерпретировать последующие символы. В тех случаях, когда ANSI-драйвер не загружен, MS-DOS передает Esc-последовательность на экран точно так же, как и любой другой текст, что приводит к нелепому изображению на экране.

Esc-последовательности, допустимые при работе с ANSI-драйвером для управления экраном, являются подмножеством последовательностей, определяемым стандартом ANSI 3.64–2979. Эти стандартные последовательности представлены на рис. 6.2.

Escape-последовательность	**Описание**
Esc [2J	Очистить экран; установить курсор в верхний левый угол (начальная позиция)
Esc [K	Стереть символы от текущей позиции курсора до конца строки
Esc [*row*; *col*H	Установить курсор на заданную позицию (*Row* – координата *y* в диапазоне 1–25 и *col* – координата *x* в диапазоне 1–80 при работе в текстовом режиме 80 × 25). Esc-последовательности, которые завершаются буквой *f* вместо *H*, выполняют те же действия
Esc [*n*A	Сдвиг курсора на *n* строк вверх
Esc [*n*B	Сдвиг курсора на *n* строк вниз
Esc [*n*C	Сдвиг курсора на *n* позиций вправо
Esc [*n*D	Сдвиг курсора на *n* позиций влево
Esc [s	Сохранение текущей позиции курсора
Esc [u	Восстановление сохраненной позиции курсора
Esc [6n	Возвращение текущей позиции курсора в формате Esc [*row*; *col*R через дескриптор стандартного ввода
Esc [*n*m	Установка атрибутов символов: 0 – отмена всех атрибутов 1 – повышенная яркость 2 – пониженная яркость 3 – курсив 4 – подчеркивание 5 – мерцание 6 – ускоренное мерцание 7 – инвертирование изображения 8 – сделать текст «невидимым» (не отображать на экране дисплея) 30 – цвет символа черный 31 – цвет символа красный 32 – цвет символа зеленый 33 – цвет символа коричневый 34 – цвет символа синий 35 – цвет символа фиолетовый 36 – цвет символа бирюзовый 37 – цвет символа белый 40 – цвет фона черный 41 – цвет фона красный 42 – цвет фона зеленый 43 – цвет фона коричневый 44 – цвет фона синий

Рис. 6.2. Управляющие Esc – последовательности стандарта ANSI, поддерживаемые системным драйвером ANSI.SYS.

Если ANSI.SYS загружен, программы, выполняющиеся в среде системы MS-DOS версии 2.0 и старше, могут использовать эти функции для управления изображением на экране дисплея вне зависимости от аппаратных средств. Символ ESC обозначает специальный ASII-код Escape, символ с кодом IBH. Заметим, что с помощью управляющих Esc-последовательностей стандарта ANSI положения курсора задаются с 1, а ПЗУ BIOS – с 0. В управляющей Esc-последовательности числа должны быть представлены в виде строки цифр в коде ASII, а не в виде своих двоичных значений.

	45 – цвет фона фиолетовый 46 – цвет фона бирюзовый 47 – цвет фона белый
Esc [-*n*h	Установка режима экрана: 0 – текстовый 40 × 25, 16 цветов (смешивание цветов выключено) 1 – текстовый 40 × 25, 16 цветов 2 – текстовый 80 × 25, 16 цветов (смешивание цветов выключено) 3 – текстовый 80 × 25, 16 цветов 4 – графический 320 × 200, 4 цвета 5 – графический 320 × 200, 4 цвета (смешивание цветов выключено) 6 – графический 620 × 200, 2 цвета 14 – графический 640 × 200, 16 цветов (EGA и VGA, MS-DOS 4.0) 15 – графический 640 × 350, 2 цвета (EGA и VGA, MS-DOS 4.0) 16 – графический 640 × 350, 16 цветов (EGA и VGA, MS-DOS 4.0) 17 – графический 640 × 480, 2 цвета (MCGA и VGA, MS-DOS 4.0) 18 – графический 640 × 480, 16 цветов (VGA, MS-DOS 4.0) 19 – графический 320 × 200, 256 цветов (MCGA и VGA, MS-DOS 4.0) Esc-последовательности, которые завершаются буквой *l* вместо *h*, выполняют те же действия.
Esc [= 7h	Перенос конца длинной строки на следующую строку экрана
Esc [= 7l	Выключение переноса конца строки, выходящей за пределы экрана

Рис. 6.2.

Заметим, что регистр клавиатуры (верхний/нижний) имеет значение для последнего символа Esc-последовательности, а числа всегда должны быть представлены в виде строки цифр в ASCII-коде, а не в виде своих двоичных значений. (Для перепрограммирования и переопределения клавиатуры может применяться специальный набор Esc-последовательностей, поддерживаемый драйвером ANSI.SYS, но несовместимый со стандартом ANSI.)

Режим двоичного вывода

В системе MS-DOS версии 2 и старше для корректных прикладных программ можно, выбрав двоичный режим работы стандартного вывода, существенно увеличить скорость воспроизведения изображения на экране, не жертвуя при этом аппаратной независимостью. В двоичном режиме MS-DOS проверяет между символами, которые она передает, не ждет ли Ctrl-C во входном буфере. Кроме того, система фильтрует выходные строки на предмет выявления определенных символов, например комбинации Ctrl-Z.

Бит 5 информационного слова устройства, связанный с дескриптором устройства, управляет двоичным режимом. Программы обращаются к информационному слову устройства с помощью подфункций 00H и 01H IOCTL-функции MS-DOS (I/O Control, управление вводом-выводом; прерывание Int 21H с функцией 44H). Например, следующий фрагмент переводит дескриптор стандартного вывода в двоичный режим.

```
                        ; Получить информацию об устройстве
mov     bx,1            ; Дескриптор стандартного вывода
mov     ax,4400h        ; Подфункция 00h функции 44h
int     21h             ; Переход в MS-DOS
```

```
mov     dh,0        ; Установить старший байт DX = 0
or      dl,20h      ; Установить бит двоичного режима в DL

                    ; Записать информацию об устройстве
                    ; ( в BX по-прежнему дескриптор )
mov     ax,4401h    ; Подфункция 00h функции 44h
int     21h         ; Переход в MS-DOS
```

Отметим, что если программа изменяет режим какого-нибудь стандартного дескриптора, она обязана восстановить ASCII-режим этого дескриптора перед завершением. В противном случае последующие прикладные программы могут повести себя непредсказуемым образом. Более детальную информацию о IOCTL-функции см. в разд. 2 «Справочник по функциям MS-DOS».

ВИДЕОФУНКЦИИ ПЗУ BIOS

Применяя видеодрайвер ПЗУ BIOS вместо функций вывода MS-DOS, можно несколько улучшить производительность интерактивных программ, предназначенных для выполнения лишь на машинах, совместимых с IBM PC. Доступный с помощью прерывания Int 10H драйвер ПЗУ BIOS поддерживает для всех известных в настоящее время видеоадаптеров фирмы IBM следующие функции:

Функция	**Описание**
Управление видеорежимом	
00H	Установить видеорежим
0FH	Получить видеорежим
Управление курсором	
01H	Установить тип курсора
02H	Установить позицию курсора
03H	Получить позицию курсора и размер
Запись изображения	
09H	Записать символ и атрибут в позицию курсора
0AH	Записать символ в позицию курсора
0EH	Записать символ в режиме телетайпа
Считывание изображения	
08H	Прочитать символ и атрибут в позицию курсора
Графическая поддержка	
0CH	Записать графический пиксел
0DH	Прочитать графический пиксел

Прокрутка или очистка изображения

06H	Инициализировать или прокрутить окно вверх
07H	Инициализировать или прокрутить окно вниз

Разное

04H	Прочитать позицию светового пера
05H	Установить видеостраницу
0BH	Установить палитру, фон или край экрана

Имеющиеся в ПЗУ BIOS дополнительные функции для видеоплат EGA, MCGA, VGA и PCjr служат для поддержки улучшенных свойств этих адаптеров, таких, как программируемая палитра и символьные наборы (шрифты). Некоторые из функций допустимы только в определенных видеорежимах.

Все видеорежимы характеризуются числом цветов, которые они могут воспроизводить, вертикальным и горизонтальным разрешением, а также, поддерживают ли текстовый или графический режим отображения на память. ПЗУ BIOS идентифицирует видеорежимы с помощью уникальных номеров. Все видеорежимы и функции прерывания Int 10H ПЗУ BIOS представлены в разд. 3 «Справочник по функциям ПЗУ BIOS и манипулятора мышь».

Как можно видеть из предыдущей таблицы, ПЗУ BIOS дает несколько полезных возможностей, которых нет в MS-DOS, включая инициализацию или прокрутку выбранных экранных окон, модификацию вида курсора, считывание назад символа, отображаемого в любом месте экрана. Эти функции позволяют отличить вашу программу от аппаратных средств любого адаптера, совместимого с IBM PC. Однако функции ПЗУ BIOS не удовлетворяют всем требованиям быстродействующих интерактивных экранных программ, таким, как текстовый процессор. Они не поддерживают быстрое отображение символьных строк в произвольной позиции экрана, не выполняют графические операции на том уровне, который обычно требуется прикладным программам (например, операции пересылки строк битов, быстрого изображения линий, окружностей и закрашенных многоугольников). И разумеется, они никак не могут использоваться в видеорежимах, отличающихся от стандартов фирмы IBM, как, например, в монохромном графическом режиме графической платы Hercules Graphics Card.

Рассмотрим простой пример вызова видеодрайвера ПЗУ BIOS. Следующий фрагмент выводит строку hello на экран дисплея:

```
msg     db      'hello'
msg-len equ     $-msg
        .
        .
        .
        mov     si,seg msg  ; DS:SI = адрес сообщения
        mov     ds,si
        mov     si,offset msg
        mov     cx,msg-len  ; CX = длина сообщения
        cld
next:   lodsb               ; Получить AL = следующий символ
        push    si          ; Сохранить указатель сообщения
        mov     ah,0eh      ; Прерывание int 10h с функцией 0eh = запись
                            ; Символ в режиме телетайпа
        mov     bh,0        ; Выберем видеостраницу 0
        mov     bl,color    ; (используется только в графическом режиме)
```

```
int     10h         ; Переход в ПЗУ BIOS
pop     si          ; Восстановить указатель сообщения
loop    next        ; Выполнить цикл до исчерпания сообщения
.
.
.
```

(Отметим, что содержимое регистров S1 и D1 не сохраняется после отработки видеофункции ПЗУ BIOS.)

ТЕХНИКА ВЫВОДА ИЗОБРАЖЕНИЯ ПРИ ОБРАЩЕНИИ ПО АДРЕСАМ, ОТОБРАЖАЕМЫМ НА ОБЩЕЕ С ПАМЯТЬЮ АДРЕСНОЕ ПРОСТРАНСТВО

В тех случаях, когда прикладная программа берет на себя полностью управление видеоадаптером и буфером изображения, производительность дисплея становится максимальной. Вследствие того что буфер дисплея отображается на память, скорость, с которой символы могут выводиться на экран, ограничена только возможностью центрального процессора копировать байты из одной ячейки памяти в другую. Платой за высокую производительность является сильная зависимость от возможностей аппаратуры, причем не всегда функции выполняются должным образом на совместимых с IBM машинах или даже на новых моделях видеоадаптеров фирмы IBM.

Текстовый режим

Непосредственное программирование совместимых с IBM PC видеоадаптеров в текстовых видеорежимах (иногда также называемых алфавитно-цифровыми) несложно. Символьный набор стандартизован, начальная позиция курсора – $(x, y) = (0, 0)$ – определена так, чтобы она находилась в верхнем левом углу экрана (рис. 6.3). Адаптер MDA использует в качестве буфера регенерации 4 Кбайта памяти, начиная с адреса B000H, а другие адаптеры, обладающие как текстовыми, так и графическими возможностями (CGA, EGA, MCGA и VGA), используют 16 Кбайт памяти, начиная с адреса B800H (рис. 6.1). В этом случае 16 Кбайт памяти разбиваются на «страницы», которые обновляются и воспроизводятся независимо друг от друга.

На каждую позицию экрана для изображения символа в буфере регенерации отводится 2 байт. Первый байт (четный адрес) содержит ASCII-код символа, который с помощью специального знакогенератора преобразуется в точечный образ символа на экране. Второй байт (нечетный адрес) – это байт атрибута. Несколько бит этого байта в зависимости от типа адаптера и видеорежима (рис. 6.4 и 6.5) управляют такими характеристиками, как мерцание, яркость (подсветка), инверсное изображение. Рис. 6.6 демонстрирует шестнадцатеричный и ASCII-дамп части видеобуфера адаптера MDA.

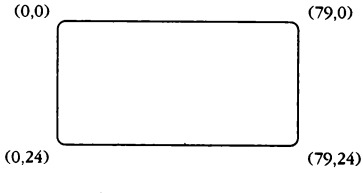
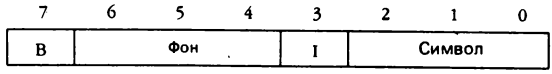


Рис. 6.3. Адресация курсора при работе в текстовых режимах 80 × 25 (режимы 2, 3 и 7 ПЗУ BIOS).

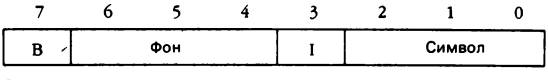


B = мерцание
I = яркость

Изображение	Фон	Символ
Невидимое (черное)	000	000
Невидимое (белое)[1)]	111	111
Подчеркнутое	000	001
Нормальное	000	111
Инверсное	111	000

[1)] Только для адаптера VGA.

Рис. 6.4. Байт атрибута при работе в монохромном текстовом режиме 80 × 25 для адаптеров MDA, Hercules cards, EGA и VGA (режим 7 ПЗУ BIOS).



B — мерцание или яркость фона (по умолчанию — мерцание)
I — яркость изображения символа или выбор символа (по умолчанию — яркость)

Значение	Цвет
0	Черный
1	Синий
2	Зеленый
3	Бирюзовый
4	Красный
5	Фиолетовый
6	Коричневый
7	Белый
8	Серый
9	Голубой
10	Салатовый
11	Светло-бирюзовый
12	Розовый
13	Светло-фиолетовый
14	Желтый
15	Ярко-белый

Рис. 6.5. Байт атрибута при работе в текстовом режиме 40 × 25 и 80 × 25 для адаптеров CGA, EGA, MCGA и VGA (режимы 0–3 ПЗУ BIOS).

По умолчанию предполагается, что таблица значений цветов палитры программируется, причем биты B и I управляют яркостью.

Вы можете вычислить относительный адрес в памяти для любого символа, выводимого на экран дисплея, если номер строки (координата *y*) умножите на 80 (символов в строке) и умножите на 2 (байт на каждый символ), прибавите номер столбца (координата *x*) и результат умножите на 2 (байт на каждый символ) и прибавите к результату (для текстографических адаптеров) произведение номера страницы на размер страницы (в режимах 80 × 25 – 4 Кбайт на страницу, а в режимах 40 × 25 – 2 Кбайт на страницу). Короче говоря, формула относительного адреса пары

```
B000:0000 3e 07 73 07 65 07 6c 07 65 07 63 07 74 07 20 07
B000:0010 74 07 65 07 6d 07 70 07 20 07 20 07 20 07 20 07
B000:0020 20 07 20 07 20 07 20 07 20 07 20 07 20 07 20 07
B000:0030 20 07 20 07 20 07 20 07 20 07 20 07 20 07 20 07
B000:0040 20 07 20 07 20 07 20 07 20 07 20 07 20 07 20 07
B000:0050 20 07 20 07 20 07 20 07 20 07 20 07 20 07 20 07
B000:0060 20 07 20 07 20 07 20 07 20 07 20 07 20 07 20 07
B000:0070 20 07 20 07 20 07 20 07 20 07 20 07 20 07 20 07
B000:0080 20 07 20 07 20 07 20 07 20 07 20 07 20 07 20 07
B000:0090 20 07 20 07 20 07 20 07 20 07 20 07 20 07 20 07
```

Рис. 6.6. Пример дампа первых 160 байт буфера регенерации, относящихся к первой строке, видимой на экране, для адаптера MDA.

Отметим, что ASCII-коды символов хранятся в четных байтах, а соответствующие им атрибуты – в нечетных; все символы строки этого примера имеют атрибут обычного видеоизображения.

символ–атрибут для заданной позиции экрана (x, y) в текстовом режиме 80 × 25 имеет вид

относительный адрес = ((y * 50H + x) * 2) + (страница * 1000H)

В текстовом режиме 40 * 25 формула выглядит так:

относительный адрес = ((y * 50H + x) * 2) + (страница * 0800H)

Разумеется, используемый для адресации видеобуфера сегментный регистр должен устанавливаться в соответствии с типом видеоадаптера.

Для примера предположим, что символ, который выводится на экран дисплея, находится в регистре AL, байт его желательного атрибута – в регистре AH, координата (столбец) x – в регистре BX, координата (строка) y – в регистре CX. Приведенная ниже программа записывает символ и байт атрибута в буфер видеоизображения адаптера MDA в соответствующее место:

```
push ax             ; Сохранить символ и атрибут
mov  ax,160
mul  cx             ; DX:AX = Y * 160
shl  bx,1           ; Умножить X на 2
add  bx,ax          ; BX = (Y*160) + (X*2)
mov  ax,0b000h      ; ES = сегмент буфера регенерации
mov  es,ax          ; монохромного адаптера
pop  ax             ; Восстановить символ и атрибут
mov  es:[bx],ax     ; Записать их значения в видеобуфер
```

Чаще нам хотелось бы перенести в буфер регенерации целую строку, начиная с заданной координаты. В следующем примере предполагается, что регистры DS:SI указывают на исходную строку, регистры ES:DI – на начальную позицию в видеобуфере (которая вычислена так же, как и в предыдущем примере), регистр AH содержит байт атрибута, который присваивается всем символам строки, регистр CX – длину строки. Далее в программе вся строка символов переносится в буфер регенерации:

```
xfer:   lodsb               ; Получить следующий символ
        stosw               ; Сохранить символ + атрибут
        loop    xfer        ; до исчерпания всех символов
```

Очевидно, что видеодрайверы, написанные для реальных прикладных программ, должны учитывать многие дополнительные факторы типа проверки специальных управляющих кодов (перевод строки, возврат каретки, табуляция), автоматического перехода на следующую строку и прокрутки.

Программы, которые записывают символы прямо в буфер изображения адаптера CGA в текстовых режимах, имеют дело с дополнительным осложняющим фактором: им необходимо проверять порт состояния видеоконтроллера и обращаться к буферу регенерации только во время интервалов горизонтального или вертикального обратного хода развертки. (Интервал обратного хода развертки – это период, в течение которого электронный пучок, облучающий люминофор экрана, возвращается к началу новой строки развертки.) В противном случае спор за память между центральным процессором и видеоконтроллером проявляется в виде неприглядного «снега» на экране. (Если вы пишите программы для любого другого видеоадаптера, совместимого с IBM PC, типа MDA, EGA, MCGA или VGA, то можете игнорировать интервалы обратного хода развертки: для этих видеоконтроллеров «снег» не является проблемой.)

Следя за определенными битами регистра состояния видеоконтроллера, программа может обнаружить наступление интервала обратного хода развертки. Например, пусть относительный адрес положения требуемого символа, вычисленный так же, как и в предыдущем примере, помещен в регистр BX, сегментный адрес буфера регенерации адаптера CGA указан в регистре ES, а в регистре CL – ASCII-код символа, который будет выведен на экран дисплея. Следующая программа ждет начала нового интервала обратного хода развертки по горизонтали, а затем записывает символ в буфер:

```
        mov     dx,03dah    ; DX = адрес порта состояния
                            ; видеоконтроллера
        cli                 ; Запретить прерывания

                            ; Ожидание окончания обратного хода,
                            ; если он уже идет ...
                            ;
wait1:  in      al,dx       ; Прочитать порт состояния
        and     al,1        ; Проверить установление бита обратного хода
        jnz     wait1       ; Если установлен, то ждать

                            ; Ожидание начала следующего интервала
                            ; обратного хода  ...
wait2:  in      al,dx       ; Прочитать порт состояния
        and     al,1        ; Бит обратного хода уже установлен?
        jz      wait2       ; Перейти, если еще нет

        mov     es:[bx],cl  ; Записать символ в буфер
                            ; регенерации
        sti                 ; Разрешить вновь прерывания
```

Первый цикл ожидания «синхронизует» программу с началом интервала обратного хода развертки по горизонтали. Если использовать только второй цикл ожидания, т. е. если символ записывается во время интервала обратного хода, запись может начинаться так близко к концу «окна» обратного хода развертки по горизонтали, что интервал будет частично перекрыт, а это приведет к «выпадению снега» на левом краю дисплея. Заметим, что программа блокирует на время обращения к видеобуферу все прерывания, таким образом, обслуживание аппаратных прерываний не будет мешать процессу синхронизации.

Из-за ограничений, связанных с длительностью интервала обратного хода разверт-

ки, скорость, с которой можно обновлять адаптер CGA в текстовом режиме, сильно ограничена, если эту операцию проводить по одному символу. Можно достичь значительно лучших результатов тем, что вычислить все необходимые адреса, установить соответствующие регистры, заблокировать видеоконтроллер путем записи в регистр 3D8H и перенести в буфер целиком строку символов с помощью операции REP MOVSV, а затем вновь разблокировать видеоконтроллер. Если строка имеет разумную длину, то пользователь даже не заметит нежелательных мерцаний на экране дисплея. Разумеется, эта процедура вносит в программу дополнительную аппаратную зависимость, так как она требует значительно больших сведений о контроллере 6845. К счастью, в графических режимах адаптера CGA проблемы «снега» нет.

Графический режим

Программирование совместимых с IBM PC адаптеров в графическом режиме путем обращения по адресам, отображаемым на общее с памятью адресное пространство, значительно сложнее программирования в текстовом режиме. Каждый бит или группа битов буфера регенерации соответствуют адресуемой точке, или пикселу, на экране дисплея. Отображение битов на пикселы различно для каждого из графических режимов в зависимости от их разрешения и числа поддерживаемых цветов. В новых адаптерах (EGA, MCGA и VGA) также применяется концепция битовых плоскостей, когда биты пиксела распределяются по нескольким банкам памяти, отображаемым по одним и тем же адресам. При работе с битовыми плоскостями следует комбинировать ввод-вывод с отображением на память и адресацию к портам.

Такой предмет, как программирование графических видеосистем фирмы IBM, заслуживает отдельной книги. Однако мы все же проиллюстрируем работу адаптера CGA, который поддерживается всеми последующими текстографическими адаптерами фирмы IBM, в графическом режиме 640 × 200, 2 цвета. С этим режимом иметь дело достаточно просто, так как здесь каждый пиксел представлен отдельным битом. Пикселам назначены координаты (x, y) в области от (0, 0) до (639, 199), где x – горизонтальное смещение, y – вертикальное смещение, а начальная позиция (0, 0) находится в верхнем левом углу экрана дисплея (рис. 6.7).

Каждая последовательная группа из 80 байт (640 бит) представляет одну строку горизонтальной развертки. Внутри каждого байта биты отображаются один к одному на пикселы, причем старший бит соответствует самому левому пикселу набора из восьми пикселов, а младший бит – самому правому пикселу того же набора. Карта отображения памяти предусматривает, что все четные координаты y входят в один набор, а все нечетные координаты y – в другой. Такое отображение называется «чередование памяти».

Для нахождения относительного адреса данной координаты (x, y) в буфере регенерации необходимо применить следующую формулу:

относительный адрес = ((y И 1) * 2000H) + (y/2 * 50H) + (x/8)

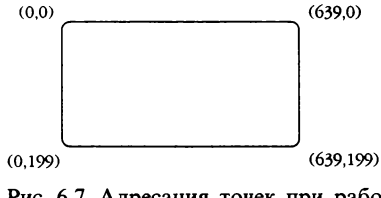


Рис. 6.7. Адресация точек при работе в графическом режиме 640 × 200, 2 цвета, для адаптеров CGA, EGA, MCGA и VGA (режим 6 ПЗУ BIOS).

В программе на языке ассемблера реализация этой формулы выглядит следующим образом:

```
                        ; Выберем AX = Y, BX = X
        shr     bx,1    ; Разделить X на 8
        shr     bx,1
        shr     bx,1
        push    ah      ; Сохранить копию Y
        shr     ax,1    ; Найти произведение (Y/2) * 50h и
        mov     cx,50h  ; поместить его в DX:AX
        mul     cx
        add     bx,ax   ; Добавить произведение к X/8
        pop     ax      ; Добавить (Y И 1) * 2000h
        and     ax,1
        jz      label1
        add     bx,2000h
label1:                 ; BX = относительный адрес
                        ; в видеобуфере
```

После вычисления адреса байта можно для вычисления положения бита заданной координаты пиксела применить другую формулу:

$$\text{бит} = 7 - (x \text{ MOD } 8),$$

где бит 7 – старший, а бит 0 – младший.
Проще всего построить 8-байт таблицу или матрицу битовых масок и для выделения соответствующей записи из таблицы использовать операцию *X И 7*:

(*X И 7*)	*Битовые маски*	(*X И 7*)	*Битовые маски*
0	80H	4	08H
1	40H	5	04H
2	20H	6	02H
3	10H	7	01H

В программе на языке ассемблера выполнение этого вычисления выглядит так:

```
table   db      80h     ; X И 7 = относительный адрес 0
        db      40h     ; X И 7 = относительный адрес 1
        db      20h     ; X И 7 = относительный адрес 2
        db      10h     ; X И 7 = относительный адрес 3
        db      08h     ; X И 7 = относительный адрес 4
        db      04h     ; X И 7 = относительный адрес 5
        db      02h     ; X И 7 = относительный адрес 6
        db      01h     ; X И 7 = относительный адрес 7
        .
        .
        .
                        ; Выберем BX = координата X
        and     bx,7    ; Выделить относительный адрес 0-7
        mov     al,[bx+table]
                        ; Теперь AL = маска из таблицы
```

Далее для установки или очистки соответствующего бита в буфере регенерации

видеоконтроллера программа может использовать маску вместе с предварительно вычисленным относительным адресом байта.

ГЛАВА 7. ПРИНТЕР И ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ

Система MS-DOS поддерживает работу принтеров, плоттеров, модемов, а также других устройств, служащих для получения документальных копий и для обеспечения связи. Поддержка осуществляется с помощью драйверов параллельных и последовательных портов. Параллельные порты называются так потому, что они передают байт (8 бит) информации устройству-адресату одновременно по восьми отдельным физическим путям (вместе с дополнительными сигналами состояния и квитирования). С другой стороны, последовательный порт сообщается с ЦП побайтно, но посылает данные своему устройству-адресату или получает данные от него последовательно, бит за битом, по единственной физической линии.

Обычно параллельные порты применяются для устройств ускоренного вывода типа построчно печатающего устройства, при передаче данных на относительно малые расстояния (менее 15 м). Они редко используются для устройств, требующих дуплексной (двусторонней) связи с компьютером. Последовательные порты используются для медленных устройств типа модема и терминала, которые требуют дуплексной связи (хотя некоторые принтеры имеют также последовательные интерфейсы). Последовательный порт может надежно управлять устройством на значительно больших расстояниях (до 300 м) всего лишь по трем линиям: передачи, приема и нулевой.

Чаще всего последовательный интерфейс следует стандарту под названием RS-232. Стандарт определяет взаимодействие устройств с помощью 25 линий с заданными электрическими характеристиками, различных сигналов квитирования и стандартного разъема DB-25. Существуют и другие стандарты последовательного интерфейса, например стандарт RS-422, обладающий значительно большей скоростью, чем RS-232, однако в настоящее время они применяются в персональных компьютерах редко (исключение составляют компьютеры Apple Macintosh).

Система MS-DOS включает встроенные драйверы устройств с тремя параллельными и двумя последовательными адаптерами на машинах типа PC и PC/AT, а на машинах PS/2–с тремя последовательными адаптерами. Эти устройства имеют логические имена LPT1, LPT2, LPT3, COM1, COM2 и COM3. За стандартным принтером (PRN) и стандартным дополнительным устройством (AUX) обычно закреплены имена LPT1 и COM1, однако вы можете перенаправить PRN на один из последовательных портов с помощью команды MS-DOS MODE.

Как и в случае ввода с клавиатуры и вывода на экран, управлять принтером и последовательным портом можно на нескольких уровнях, отличающихся разной степенью гибкости и аппаратурной независимости:

- Функции типа дескриптора системы MS-DOS
- Традиционные символьно-ориентированные функции системы MS-DOS
- Функции драйвера ПЗУ BIOS

В случае последовательного порта непосредственное управление аппаратурой из прикладных программ осуществляется традиционно. На следующих страницах кратко обсуждается каждый из этих методов ввода-вывода и приводятся примеры.

ВЫВОД НА ПРИНТЕР

Для вывода на принтер предпочтительно использовать функцию записи типа дескриптора (прерывание Int 21H с функцией 40H) с предопределенным дескриптором

(4) стандартного принтера. Например, строку с текстом hello можно вывести на принтер следующим образом:

```
msg_len equ     $-msg       ; Длина сообщения
        .
        .
        .
        mov     ah,40h      ; Функция 40h = запись файла или устройства
        mov     bx,4        ; BX = дескриптор стандартного принтера
        mov     cx,msg_len  ; CX = длина строки
        mov     dx,seg msg  ; DS:DX = адрес строки
        mov     ds,dx
        mov     dx,offset msg
        int  21h            ; Переход в MS-DOS
        jc      error       ; Перейти, если ошибка
        .
        .
        .
```

Если при этом не произойдет ошибки, то функция сбросит флаг переноса и поместит в регистр AX число символов, фактически переданных на устройство печати. В нормальных ситуациях значение этого числа должно равняться затребованной длине строки, а флаг переноса, указывающий на ошибку, никогда не должен быть установлен. Однако если в данных содержится признак конца файла (Ctrl-Z), то вывод завершится раньше.

Можно осуществить вывод независимо на несколько устройств печати (например, LPT1 и LPT2) путем использования для каждого из них запроса на операцию открытия и применения возвращенных дескрипторов при индивидуальном обращении к принтерам по прерыванию Int 21H с функцией 40H. Этот общий подход уже демонстрировался в гл. 5 и 6.

Альтернативный метод вывода на принтер заключается в использовании традиционного обращения по прерыванию Int 21H с функцией 05H, которое передает на принтер символ из регистра DL. (Эта функция «чувствует» прерывания по комбинации Ctrl-C.) Например, следующий фрагмент программы на языке ассемблера записывает строку hello на устройство вывода:

```
msg     db      'hello'     ; Сообщение на принтер
msg_len equ     $-msg       ; Длина сообщения
        .
        .
        .
        mov     bx,seg msg  ; DS:BX = адрес строки
        mov     ds,bx
        mov     bx,offset msg
        mov     cx,msg_len  ; CX = длина строки

next:   mov     dl,[bx]     ; Получить следующий символ
        mov     ah,5        ; Функция 05h = вывод на принтер
        int  21h            ; Переход в MS-DOS
        inc     bx          ; Инкремент указателя строки
        loop    next        ; Выполнить цикл до исчерпания строки
        .
        .
        .
```

Программы, которые выполняются на машинах, совместимых с IBM PC, могут достичь лучшей производительности печати путем обхода MS-DOS и обращения с помощью прерывания Int 17H ПЗУ BIOS непосредственно к драйверу принтера. В разд. 3 «Справочник по функциям ПЗУ BIOS и манипулятора мышь» содержится подробная информация о прерывании Int 21H и его функциях. Применение функций ПЗУ BIOS позволяет также программе проверять, включен ли принтер и есть ли в нем бумага, т. е. дает возможности, которые не предоставляют функции MS-DOS.

Например, следующий фрагмент для вывода строки hello на устройство печати вызывает функции ПЗУ BIOS управления драйвером принтера:

```
msg     db      'hello'     ; Сообщение на принтер
msg_len equ     $-msg       ; Длина сообщения
        .
        .
        .
        mov     bx,seg msg  ; DS:BX = адрес строки
        mov     ds,bx
        mov     bx,offset msg
        mov     cx,msg_len  ; CX = длина строки
        mov     dx,0        ; DX = номер принтера

next:   mov     al,[bx]     ; AL = символ для вывода на принтер
        mov     ah,0        ; Функция 00h = вывод на принтер
        int  17h            ; Переход в ПЗУ BIOS
        inc     bx          ; Инкремент указателя строки
        loop    next        ; Выполнить цикл до исчерпания строки
        .
        .
        .
```

Заметим, функции ПЗУ BIOS используют нумерацию принтеров с 0, тогда как в именах логических устройств функций принтеры нумеруются с 1. Например, 0 в функции ПЗУ BIOS управления принтером соответствует LPT1.

И наконец, непосредственное обращение к контроллеру принтера – это наиболее аппаратно-зависимый метод вывода на принтер. Принимая во внимание функциональные возможности, уже предусмотренные функциями MS-DOS и ПЗУ BIOS, а также скорость рассматриваемых устройств, никак нельзя оправдать применения в этом случае непосредственного управления аппаратурой. Неудобства от введения сильной зависимости от аппаратных средств при работе с медленными устройствами будут значительно больше, чем тот небольшой выигрыш в производительности, который можно получить таким образом.

ПОСЛЕДОВАТЕЛЬНЫЙ ПОРТ

Система MS-DOS значительно слабее поддерживает последовательные порты (часто именуемые в руководствах по MS-DOS как дополнительные устройства) по сравнению с клавиатурой, видеосистемой и принтером. Это одно из мест, где в целях достижения адекватной производительности этих устройств оправдано создание прикладным программистом собственных программ, ориентированных на конкретную аппаратуру.

В программах, где для обеспечения их переносимости ограничиваются применением только функций MS-DOS, при обращении к последовательному порту можно использовать функции чтения и записи типа дескриптора с предопределенным дескриптором

(3) стандартного дополнительного устройства. Например, следующая программа записывает строку hello в последовательный порт, определенный здесь как дополнительное устройство:

```
msg     db      'hello'     ; Сообщение для передачи
                            ; в последовательный порт
msg_len equ     $-msg       ; Длина сообщения
        .
        .
        .
        mov     ah,40h      ; Функция 40h = запись файла или устройства
        mov     bx,3        ; BX = дескриптор стандартного
                            ; дополнительного устройства
        mov     cx,msg_len  ; CX = длина строки
        mov     dx,seg msg  ; DS:DX = адрес строки
        mov     ds,dx
        mov     dx,offset msg
        int     21h         ; Переход в MS-DOS
        jc      error       ; Перейти, если ошибка
        .
        .
        .
```

Дескриптор стандартного дополнительного устройства позволяет обращаться только к первому последовательному порту (COM1). Если вы хотите выполнить операцию чтения или записи при обращении к COM2 и COM3 с помощью дескриптора, то вам следует выдать запрос на открытие требуемого последовательного порта (прерывание Int 21H с функцией 3DH) и использовать полученный после этой операции дескриптор для осуществления требуемого действия с помощью прерывания Int 21H с функциями 3FH и 40H.

Некоторые версии MS-DOS содержат ошибку в программах обслуживания символьно-ориентированных устройств, которая проявляется следующим образом. Если вы выдаете запрос на операцию чтения с помощью прерывания Int 21H с функцией 3FH точно по числу символов, ожидающих в буфере драйвера, то возвращаемая в регистр AX длина равна числу переданных символов минус один. Это обстоятельство всегда можно обойти, запрашивая большее число символов, чем ожидается получить, или переводя дескриптор устройства в двоичный режим с помощью прерывания Int 21H с функцией 44H.

Система MS-DOS поддерживает также две традиционные функции ввода-вывода для последовательного порта. Прерывание Int 21H с функцией 03H осуществляет чтение символа из COM1 и возврат его в регистре AL в вызывающую программу. Прерывание Int 21H с функцией 04H осуществляет передачу символа из регистра DL на COM1. Как и другие традиционные обращения, эти две функции – прямые потомки функций дополнительного устройства системы CP/M.

В качестве примера приведем программу пересылки символьной строки hello в COM1 путем традиционного обращения с помощью прерывания Int 21H с функцией 04H:

```
msg     db      'hello'     ; Сообщение для передачи
                            ; в последовательный порт
msg_len equ     $-msg       ; Длина сообщения
        .
        .
        .
```

```
        mov     bx,seg msg  ; DS:BX = адрес строки
        mov     ds,bx
        mov     bx,offset msg
        mov     cx,msg_len  ; CX = длина строки

next:   mov     dl,[bx]     ; Получить следующий символ
        mov     ah,4        ; Функция 04h = вывод в порт
        int     21h         ; Переход в MS-DOS
        inc     bx          ; Инкремент указателя строки
        loop    next        ; Выполнить цикл до исчерпания строки
        .
        .
        .
```

Система MS-DOS преобразует традиционные функции дополнительного устройства в обращения к тому же драйверу устройства, с которым работают функции типа дескриптора. Поэтому в общем случае предпочтительнее применять функции типа дескриптора, так как они позволяют за одну операцию прочитать или записать очень длинную строку текста, обеспечивают доступ к другим, кроме COM1, последовательным портам, а также обладают симметрией относительно операций ввода-вывода с помощью дескриптора при работе с экраном, клавиатурой, принтером и файлами, описанных в других местах книги.

Хотя функции типа дескриптора и традиционные функции управления последовательным портом позволяют писать программы, переносимые на любую машину с системой MS-DOS, тем не менее они обладают рядом недостатков:

- Встроенный системный драйвер последовательного порта отличается невысокой скоростью и неуправляем по прерываниям.
- Последовательный порт выполняет системные операции ввода-вывода без буферизации.
- Определение состояния последовательного порта требует специального вызова IOCTL-функции (с помощью прерывания Int 21H с функцией 44H), причем если запросить выполнения операции ввода и не приготовить никаких символов на ввод, то программа зависнет.
- Система MS-DOS не предоставляет никаких стандартных функций конфигурирования последовательного порта прямо из программы.

Для программ, предназначенных для выполнения на IBM PC или совместимых с ними компьютерах, наиболее гибкий способ работы с последовательным коммуникационным портом состоит в обращении к драйверу последовательного порта ПЗУ BIOS по прерыванию Int 21H. Этот драйвер применяется для инициализации последовательного порта с желательной конфигурацией и скоростью передачи информации (в бодах), проверки состояния контроллера, а также выполнения операций чтения и записи символов. В разд. 3 «Справочник по функциям ПЗУ BIOS и манипулятора мышь» приведены функции ПЗУ BIOS для работы с драйвером последовательного порта.

Например, в следующей программе символ X пересылается в первый последовательный порт (COM1):

```
        .
        .
        .
        mov     ah,1        ; Функция 01h = передать символ
        mov     al,'X'      ; AL =  символ для передачи
        mov     dx,0        ; DX = номер последовательного порта
        int     14h         ; Переход в ПЗУ BIOS
        and     ah,80h      ; Передача не выполнена ?
```

```
jnz     error       ; Перейти, если ошибка передачи
.
.
.
```

Как и в случае драйвера принтера ПЗУ BIOS, функции ПЗУ BIOS используют нумерацию последовательных портов с 0, тогда как в именах логических устройств функции MS-DOS последовательные порты нумеруются с 1. В данном примере последовательный порт 0 соответствует COM1.

К сожалению, подобно драйверу дополнительного устройства системы MS-DOS, драйвер последовательного порта ПЗУ BIOS не управляется прерываниями. Хотя он будет поддерживать более высокую скорость передачи данных, чем функции MS-DOS, тем не менее при скорости более 2400 бод может произойти потеря символов. Поэтому большинство программистов, пишущих высокопроизводительные программы с обращением к последовательному порту (как, например, телекоммуникационные программы), берут на себя все управление контроллером последовательного порта и обеспечивают его своим собственным драйвером, управляемым по прерываниям. В случае компьютеров IBM PC встроенные функции системы MS-DOS и ПЗУ BIOS просто неадекватны.

Написание таких программ требует хорошего понимания работы аппаратных средств. В случае компьютеров IBM PC необходимо изучать микросхемы: INS8250 – контроллер асинхронной связи и Intel 8259A – программируемый контроллер прерываний. Техническая документация фирмы IBM по этим микросхемам не слишком последовательна, но большая часть необходимой информации в ней содержится (хотя поиск ее потребует определенных усилий).

ПРОГРАММА TALK

Простая программа TALK.ASM (рис. 7.1), эмулятор терминала, – это пример удачной программы, осуществляющей операции ввода-вывода через такие устройства, как экран, клавиатура и последовательный порт. В программе проиллюстрированы все вопросы, обсуждавшиеся в гл. 5–7. Программа TALK использует видеодрайвер ПЗУ BIOS для вывода символов на экран, очистки изображения и позиционирования курсора; применяет символьно-ориентированные функции MS-DOS для операции чтения с клавиатуры; кроме того, содержит свой собственный управляемый по прерываниям драйвер контроллера последовательного порта.

Программа TALK иллюстрирует приемы, которые прикладные программы должны использовать для обслуживания прерываний от последовательного порта, без нарушения соглашений системы MS-DOS.

Программа начинается с ряда присвоений и условных операторов языка ассемблера, которые настраивают программу на дуплексную (двухстороннюю) или полудуплексную (одностороннюю) связь и требуемый последовательный порт (COM1 или COM2). Сразу после получения управления от MS-DOS основная часть программы, процедура talk, проверяет состояние последовательного порта, инициализирует видеоизображение и вызывает подпрограмму asc_enb, для того чтобы принять вектор прерывания последовательного порта и разрешить прерывания. Далее процедура talk входит в цикл, в котором выполняются операции чтения с клавиатуры и передача символов в последовательный порт, а затем чтение из последовательного порта и вывод символов на экран дисплея, т. е., другими словами, процедура заставляет компьютер эмулировать работу обычного терминала.

Программа TALK перехватывает и соответственно обрабатывает управляющие

```
        name    talk
        page    55,132
        .lfcond             ; Вывести в листинг и ложные условия тоже
        title   TALK--Простой эмулятор терминала

;
; TALK.ASM--Простой эмулятор терминала компьютера IBM PC
;
; Авторские права (c) Рей Данкан 1988г.
;
; Для трансляции и компоновки программы в файл TALK.EXE выполнить:
;
;       C>MASM TALK;
;       C>LINK TALK;
;

stdin   equ     0               ; Дескриптор стандартного ввода
stdout  equ     1               ; Дескриптор стандартного вывода
stderr  equ     2               ; Дескриптор стандартной ошибки

cr      equ     0dh             ; Код ASCII возврата каретки
lf      equ     0ah             ; Код ASCII перевода строки
bsp     equ     08h             ; Код ASCII возврата на шаг
escape  equ     1bh             ; Код ASCII Escape-клавиши

dattr   equ     07h             ; Атрибут символов для использования
                                ; в режиме эмуляции

bufsiz  equ     4096            ; Размер буфера последовательного порта

echo    equ     0               ; 0 = дуплексный и -1 = полудуплексный
                                ; режимы
true    equ     -1
false   equ     0

com1    equ     true            ; Если неравно 0, использовать COM1
com2    equ     not com1        ; Если неравно 0, использовать COM2

pic_mask  equ   21h             ; Порт маски прерываний контроллера 8259
pic_eoi   equ   20h             ; Порт сигнала завершения прерывания (EOI)
                                ; контроллера 8259

        if      com1
com_data equ    03f8h           ; Закрепление портов за COM1
com_ier  equ    03f9h
com_mcr  equ    03fch
com_sts  equ    03fdh
com_int  equ    0ch             ; Номер прерывания COM1
int_mask equ    10h             ; Маска IRQ4 для контроллера 8259
        endif

        if      com2
com_data equ    02f8h           ; Закрепление портов за COM2
```

Рис. 7.1. TALK.ASM. Простая программа эмулятора терминала для IBM PC и совместимых компьютеров.

Эта программа демонстрирует применение функций системы MS-DOS и ПЗУ BIOS для работы с видеоизображением и клавиатурой, а также прямое управление последовательным связным адаптером.

```
com_ier  equ    02f9h
com_mcr  equ    02fch
com_sts  equ    02fdh
com_int  equ    0bh             ; Номер прерывания COM2
int_mask equ    08h             ; Маска IRQ3 для контроллера 8259
         endif

_TEXT    segmeht word public 'CODE'

         assume  cs:_TEXT,ds:_DATA,es:_DATA,ss:STACK

talk     proc    far             ; Точка входа из MS-DOS

         mov     ax,_DATA        ; Сделаем сегмент данных адресуемым
         mov     ds,ax
         mov     es,ax
                                 ; Инициализация дисплея для
                                 ; режима эмуляции терминала ...

         mov     ah,15           ; Получить ширину дисплея и
         int     10h             ; текущий режим дисплея
         dec     ah              ; Сохранить ширину дисплея для использования
         mov     columns,ah      ; в подпрограмме очистки экрана

         cmp     al,7            ; Установить текстовый режим дисплея
         je      talk2           ; Продолжить, если это режим 7
         cmp     al,3
         jbe     talk2           ; Продолжить, если это режимы 0-3

         mov     dx,offset msg1
         mov     cx,msg1_len
         jmp     talk6           ; Вывести сообщение об ошибке и
                                 ; выйти

talk2:   mov     bh,dattr        ; Очистить экран и установить курсор
                                 ; в начальное полохение
         call    cls

         call    asc_enb         ; Захватить вектор прерывания последова-
                                 ; тельного порта и разрешить прерывания

         mov     dx,offset msg2  ; Вывести на экран сообщение
         mov     cx,msg2_len     ; "Эмулятор терминала запущен "
         mov     bx,stdout       ; BX = дескриптор стандартного вывода
         mov     ah,40           ; Функция 40h = запись файла или устройства
         int     21h             ; Переход в MS-DOS

talk3:   call    pc_stat         ; Символ с клавиатуры ждет?
         jz      talk4           ; Перейти, если символ не ждет

         call    pc_in           ; Прочитать символ с клавиатуры

         cmp     al,0            ; Это функциональная клавиша ?
         jne     talk32          ; Перейти, если нет

         call    pc_in           ; Если да, то отбросить 2-ой символ
                                 ; последовательности,
```

Рис. 7.1.

```
        jmp     talk5           ; затем завершить программу

talk32:                         ; Принятие символа с клавиатуры
        if      echo
        push    ax              ; Если режим полудуплексный,то
        call    pc_out          ; вывести символ на экран дисплея (эхо)
        pop     ax
        endif

        call    com_out         ; Записать символ в последовательный порт

talk4:  call    com_stat        ; Символ из последовательного порта ждет?
        jz      talk3           ; Перейти, если символ не ждет

        call    com_in          ; Прочитать символ из последовательного порта

        cmp     al,20h          ; Это управляющий код?
        jae     talk45          ; Перейти, если нет
        call    ctrl_code       ; Если да, то отбросить его

        jmp     talk3           ; Проверить клавиатуру еще раз

talk45:                         ; Принятие кода неуправляющего символа,
        call    pc_out          ; вывод его на экран дисплея

        jmp     talk4           ; Посмотрим, нет ли еще ожидающих символов

talk5:                          ; Обнаружение функциональной клавиши,
                                ; приготовление к завершению ...

        mov     bh,07h          ; Очистить экран и установить курсор
        call    cls             ; в начальное положение

        mov     dx,offset msg3  ; Вывести сообщение об окончании работы
        mov     cx,msg3_len

talk6:  push    dx              ; Сохранить адрес сообщения
        push    cx              ; и его длину

        call    asc_dsb         ; Запретить прерывания последовательного
                                ; порта и освободить вектор прерывания

        pop     cx              ; Восстановить длину сообщения
        pop     dx              ; и адрес

        mov     bx,stdout       ; Дескриптор стандартного вывода
        mov     ah,40h          ; Функция 40h = запись устройства
        int     21h             ; Переход в MS-DOS

        mov     ax,4c00h        ; Завершить программу
        int     21h             ; с кодом завершения = 0

talk    endp

com_stat proc   near            ; Проверка состояния кольцевого буфера;
                                ; Возвращает: Z = "ложь",если символ ждет
                                ; Z = "истина",если символ не ждет
```

Рис. 7.1.

```
        push    ax
        mov     ax,asc_in       ; Сравнить указатели кольцевого буфера
        cmp     ax,asc_out
        pop     ax
        ret                     ; Возврат в место вызова
com_stat endp

com_in proc     near            ; Получение символа из буфера последо-
                                ; вательного порта. Возврат кода
                                ; нового символа в AL

        push    bx              ; Сохранить регистр BX

com_in1:                        ; Если символ не ждет, ждать приема
        mov     bx,asc_out      ; символа
        cmp     bx,asc_in
        je      com_in1         ; Переход, символ не ждет

        mov     al,[bx+asc_buf] ; Символ готов,
                                ; извлечь его из буфера

        inc     bx              ; Обновить указатель буфера
        cmp     bx,bufsiz
        jne     com_in2
        xor     bx,bx           ; Сбросить указатель, если конец буфера
com_in2:
        mov     asc_out,bx      ; Запомнить обновленный указатель
        pop     bx              ; Восстановить регистр BX
        ret                     ; и вернуться к месту вызова

com_in  endp

com_out proc    near            ; Запись символа в регистр AL
                                ; для передачи в последовательный порт

        push    dx              ; Сохранить регистр DX
        push    ax              ; Сохранить символ для передачи
        mov     dx,com_sts      ; DX = адрес порта состояния

com_out1:                       ; Проверить, пуст ли буфер передачи
        in      al,dx           ; ( бит TBE = установлен )
        and     al,20h
        jz      com_out1        ; Нет, должны ждать

        pop     ax              ; Получить символ для передачи
        mov     dx,com_data     ; DX = адрес порта данных
        out     dx,al           ; Передать символ
        pop     dx              ; Восстановить регистр DX
        ret                     ; и вернуться к месту вызова

com_out endp

pc_stat proc    near            ; Читает состояние клавиатуры. Возвращает:
                                ; Z = "ложь", если символ готов
                                ; Z = "истина", если символ не ждет
                                ; Регистр DX не сохраняется

        mov     al,in_flag      ; Если символ уже ждет, то
```

Рис. 7.1.

```
        or      al,al           ; вернуть состояние
        jnz     pc_stat1

        mov     ah,6            ; Иначе вызвать MS-DOS для определения
        mov     dl,0ffh         ; состояния клавиатуры
        int     21h

        jz      pc_stat1        ; Перейти, если код клавиши неготов

        mov     in_char,al      ; Получить код клавиши, сохранить его
        mov     in_flag,0ffh    ; для подпрограммы pc_in

pc_stat1:                       ; Возвращение к месту вызова с установ-
        ret                     ; ленным соответствующим образом флагом Z

pc_statp

pc_in   proc    near            ; Читает символ с клавиатуры и
                                ; возвращает его в AL
                                ; Регистр DX может быть не сохранен

        mov     al,in_flag      ; Код клавиши уже ждет?
        or      al,al
        jnz     pc_in1          ; Да, вернуть его в место вызова

        call    pc_stat         ; Попытка прочитать символ
        jmp     pc_in

pc_in1: mov     in_flag,0       ; Очистить флаг ожидания символа
        mov     al,in_char      ; и вернуть значение AL = символ
        ret

pc_in   endp

pc_out  proc    near            ; Запись символа в AL
                                ; для отображения его на экране дисплея

        mov     ah,0eh          ; Функция ПЗУ BIOS 0eh =
                                ; "телетайпный вывод"
        push    bx              ; Сохранить регистр BX
        xor     bx,bx           ; Выберем страницу 0
        int     10h             ; Переход в ПЗУ BIOS
        pop     bx              ; Восстановить регистр BX
        ret                     ; и вернуться к месту вызова

pc_out  endp


cls     proc    near            ; Очистка экрана
                                ; с использованием
                                ; атрибута символа в BH
                                ; Регистры AX, CX и DX не сохраняются

        mov     dl,columns      ; Установить DL, DH = X,Y
        mov     dh,24           ; правого нижнего угла
        mov     cx,0            ; Установить CL,CH = X,Y
                                ; левого верхнего угла
        mov     ax,600h         ; Функция ПЗУ BIOS 06h =
```

Рис. 7.1.

```
                                ; прокрутка или инициализация
                                ; окна
        int     10h             ; Переход в ПЗУ BIOS
        call    home            ; Установить курсор в полохение (0,0)
        ret                     ; и вернуться к месту вызова

cls     endp

clreol  proc    near            ; Очистка экрана от текущего полохения
                                ; курсора до конца строки с использованием
                                ; атрибута символа в BH
                                ; Регистры AX, CX и DX не сохраняются

        call    getxy           ; Получить текущее полохение курсора
        mov     cx,dx           ; Текущее полохение = верхний
                                ; левый угол окна
        mov     dl,columns      ; Нижний правый угол ,X - максимальный
                                ; столбец, Y имеет то же значение,
                                ; как и в верхнем левом углу
        mov     ax,600h         ; Функция ПЗУ BIOS 06h =
                                ; прокрутка или инициализация
                                ; окна
        int     10h             ; Переход в ПЗУ BIOS
        ret                     ; Возврат к месту вызова

clreol  endp

home    proc    near            ; Перевод курсора в исходную позицию

        mov     dx,0            ; Установить (X,Y) = (0,0)
        call    gotoxy          ; Установить курсор в заданное полохение
        ret                     ; Возврат к месту вызова

home    endp

gotoxy  proc    near            ; Установка курсора в заданное полохение,
                                ; причем DX = X, DH = Y

        push    bx              ; Сохранить регистры
        push    ax

        mov     bh,0            ; Выберем страницу 0
        mov     ah,2            ; Функция ПЗУ BIOS 02h =
                                ; установить полохение курсора
        int     10h             ; Переход в ПЗУ BIOS

        pop     ax              ; Восстановить регистры
        pop     bx
        ret                     ; и возвратиться к месту вызова

gotoxy  endp


getxy   proc    near            ; Получение текущего полохения
                                ; курсора, причем DX = X, DH = Y

        push    ax              ; Сохранить регистры
```

Рис. 7.1.

```
   push    bx
   push    cx

   mov     ah,3            ; Функция ПЗУ BIOS 03h =
                           ; получить полохение курсора
   mov     bh,0            ; Выберем страницу 0
   int     10h             ; Переход в ПЗУ BIOS
         pop     cx              ; Восстановить регистры
         pop     bx
         pop     ax
         ret                     ; и вернуться к месту вызова

getxy    endp
ctrl_code proc  near             ; Обработка управляющего кода,причем
                                 ; содержимое регистра AL = символ

         cmp     al,cr           ; Если это код возврат каретки,
         je      ctrl8           ; то просто переслать его

         cmp     al,lf           ; Если это код перевод строки,
         je      ctrl8           ; то просто переслать его

         cmp     al,bsp          ; Если это код возврат на шаг,
         je      ctrl8           ; то просто переслать его


         cmp     al,26           ; Это управляющий код очистки экрана?
         jne     ctrl7           ; Нет, переход

         mov     bh,dattr        ; Управляющий код очистки экрана,
         call    cls             ; экран очистить и установить курсор
                                 ; в начальное полохение

         jmp     ctrl9

ctrl7:
         cmp     al,escape       ; Это код символа Esc?
         jne     ctrl9           ; Нет, выбросить его

         call    esc_seq         ; Да, эмулировать терминал
         jmp     ctrl9

ctrl8:   call    pc_out          ; Переслать коды возврат каретки,
                                 ; перевод строки или возврат на шаг
                                 ; на экран дисплея

ctrl9:   ret                     ; Вернуться к месту вызова

ctrl_code endp


esc_seq proc    near             ; Декодирует управляющую Esc - последо-
                                 ;  вательность Televideo 950 для
                                 ; работы с экраном

         call    com_in          ; Получить следующий символ
         cmp     al,84           ; Это код очистки до конца строки ?
         jne     esc_seq1        ; Нет, перейти
```

Рис. 7.1.

```
        mov     bh,dattr        ; Да, очистить до конца строки
        call    clreol
        jmp     esq_seq2        ; Затем выйти

esq_seq1:
        cmp     al,61           ; Это позиционирование курсора?
        jne     esq_seq2        ; Нет, перейти

        call    com_in          ; Да, получить параметр Y
        sub     al,33           ; и вычесть смещение
        mov     dh,al

        call    com_in          ; Получить параметр X
        sub     al,33           ; и вычесть смещение
        mov     dl,al
        call    gotoxy          ; Позицинировать курсор

esq_seq2:                       ; Возврат к месту вызова
        ret

esq_seq endp


asc_enb proc    near            ; Захват вектора прерывания последо-
                                ; вательного порта и разрешение прерывания

                                ; Сохранение адреса исходного
                                ; обработчика прерывания ...
        mov     ax,3500h+com_int ; Функция 35h =  получить вектор
        int     21h             ; Переход в MS-DOS
        mov     word ptr oldvec+2,es
        mov     word ptr oldvec,bx

                                ; Установка нашего обработчика...
        push    ds              ; Сохранить наш сегмент данных
        mov     ax,cs           ; Установить DS:DX = адрес
        mov     ds,ax           ; нашего обработчика прерывания
        mov     dx,offset asc_int
        mov     ax,2500h+com_int ; Функция 25h =  установить вектор
        int     21h             ; Переход в MS-DOS
        pop     ds              ; Восстановить сегмент данных

        mov     dx,com_mcr      ; Установить регистр управления модема
        mov     al,0bh          ; Биты DTR и  OUT2
        out     dx,al

        mov     dx,com_ier      ; Установить регистр разрешения
        mov     al,1            ; прерываний контроллера
        out     dx,al           ; последовательного порта
        in      al,pic_mask     ; Прочитать текущую маску контроллера 8259
        and     al,not int_mask ; Установить маску порта COM
        out     pic_mask,al     ; Записать новую маску контроллера 8259

        ret                     ; Вернуться назад к месту вызова


asc_enbp
```

Рис. 7.1.

```
asc_dsb proc    near            ; Запрет прерываний и
                                ; освобождение вектора прерываний

        in      al,pic_mask     ; Прочитать текущую маску контроллера 8259
        or      al,int_mask     ; Сбросить маску порта COM
        out     pic_mask,al     ; Записать новую маску контроллера 8259

        push    ds              ; Сохранить наш сегмент данных
        lds     dx,oldvec       ; Загрузить адрес исходного
                                ; обработчика прерывания
        mov     ax,2500h+com_int ; Функция 25h = установить вектор
        int     21h             ; Переход в MS-DOS
        pop     ds              ; Восстановить сегмент данных

        ret                     ; Вернуться назад к месту вызова

asc_dsb endp


asc_int proc    far             ; Подпрограмма обслуживания прерывания
                                ; последовательного порта

        sti                     ; Вновь включить прерывания

        push    ax              ; Сохранить регистры
        push    bx
        push    dx
        push    ds


        mov     ax,_DATA        ; Сделаем наш сегмент данных
        mov     ds,ax           ; адресуемым

        cli                     ; Очистить прерывания для
                                ; операций с указателем

        mov     dx,com_data     ; DX = адрес данных порта
        in      al,dx           ; Прочитать этот символ
        mov     bx,asc_in       ; Получить указатель буфера
        mov     [asc_buf+bx],al ; Сохранить этот символ
        inc     bx              ; Инкремент указателя
        cmp     bx,bufsiz       ; Время перейти на следующую строку ?
        jne     asc_int1        ; Нет, перейти
        xor     bx,bx           ; Да, сбросить указатель

asc_int1:                       ; Сохранение обновленного указателя
        mov     asc_in,bx

        sti                     ; Вновь включить прерывания

        mov     al,20h          ; Переслать код завершения прерывания
                                ; (EOI) в контроллер 8259
        out     pic_eoi,al

        pop     ds              ; Восстановить все регистры
        pop     dx
        pop     bx
        pop     ax
```

Рис. 7.1.

```
        iret                    ; Возврат из прерывания

asc_int endp

_TEXT   ends


_DATA   segment word public 'DATA'

in_char db      0               ; Символ с клавиатуры
in_flag db      0               ; <>0 , если символ ждет

columns db      0               ; Столбец с наибольшим номером для
                                ; текущего режима дисплея (39 или 79)

msg1    db      cr,lf
        db      'Дисплей должен быть в текстовом режиме.'
        db      cr,lf
msg_len  equ $-msg1

msg2    db      'Эмулятор терминала запущен...'
        db      cr,lf
msg2_len equ $-msg2

msg3    db      'Выход из эмулятора терминала.'
        db      cr,lf
msg3_len equ $-msg3

oldvec  dd      0               ; Исходное содержимое вектора
                                ; прерывания последовательного порта

asc_in  dw      0               ; Указатель ввода в кольцевой буфер
asc_out dw      0               ; Указатель вывода из кольцевого буфера

asc_buf db      bufsiz dup (?)  ; Коммуникационный буфер

_DATA   ends


STACK   segment para stack 'STACK'

        db      128 dup(?)

STACK   ends

        end     talk            ; Определение точки входа
```

Рис. 7.1.

коды (возврат каретки, перевод строки и т. д.). Она обнаруживает управляющие Esc-последовательности и обрабатывает их как подмножество разрешенных операций терминала Televideo 950. (Можно легко изменить программу с целью эмуляции любого другого терминала с адресуемым курсором.) Когда нажимается одна из специальных функциональных клавиш персонального компьютера, программа запрещает прерывания последовательного порта, освобождает вектор прерывания последовательного порта и выходит назад в MS-DOS.

В программе TALK имеется еще несколько процедур, которые заслуживают

внимания, так как их можно легко перенести в другие программы. Эти процедуры перечислены ниже в таблице.

Процедура	Описание
asc_enb	Перехватить вектор прерывания последовательного порта и разрешить прерывания путем записи в регистр управления модема INS8250 и регистр маски прерываний 8259A
asc_dsb	Восстановить исходное состояние вектора прерываний последовательного порта и запретить прерывания путем записи в регистр маски прерываний 8259A
asc_int	Обслужить прерывание последовательного порта, помещая полученные символы в кольцевой буфер
com_stat	Проверить наличие в кольцевом буфере ожидающих символов из последовательного порта
com_in	Удалить символы из кольцевого буфера обработчика прерывания и выполнить инкремент указателей буфера
com_out	Передать один символ в последовательный порт
cls	Вызвать видеодрайвер ПЗУ BIOS для очистки экрана
clreol	Вызвать видеодрайвер ПЗУ BIOS для очистки строки от текущей позиции курсора до конца строки
home	Поместить курсор в верхний левый угол экрана
gotoxy	Поместить курсор на требуемую позицию экрана
getxy	Получить текущее положение курсора
pc_out	Передать один символ на экран
pc_stat	Получить состояние клавиатуры
pc_in	Возвратить символ, полученный с клавиатуры

ГЛАВА 8. УПРАВЛЕНИЕ ФАЙЛАМИ

По-видимому, двойственное наследие систем CP/M и UNIX/XENIX в MS-DOS наиболее ясно прослеживается в организации управления файлами. Вообще говоря, MS-DOS предусматривает по крайней мере два разных системных вызова для всех основных операций с файлами или записями. В данной главе множество таких частично перекрывающихся функций разбивается на две группы и обсуждаются вопросы их применения, а также свойственные им преимущества и недостатки.

Набор функций управления файлами и записями, совместимых с системой CP/M, будем называть функциями типа FCB. Эти функции для получения информации об открытых файлах используют структуру данных, называемую блоком управления файлами (file control block, FCB). Эта структура постоянно хранится в пространстве памяти, выделенной прикладной программе. Функции типа FCB дают возможность программисту создавать, открывать, закрывать и уничтожать файлы, а также читать и делать записи любой длины в любом месте внутри файлов. Эти функции не поддерживают иерархическую (древовидную) файловую структуру, которая впервые введена в системе MS-DOS версии 2, поэтому ими можно пользоваться для доступа к файлам, находящимся только в текущем подкаталоге текущего диска.

Набор функций управления файлами и записями, обеспечивающих совместимость с системой UNIX/XENIX, будем называть функциями типа дескриптора. Эти функции дают возможность программисту выполнять операции открытия и создания файлов путем передачи системе MS-DOS строки символов, завершающейся нулем, в которой описываются расположение файла в иерархической файловой структуре (имя диска и путь), имя файла и его расширение. При успешном выполнении операций открытия или создания система MS-DOS возвращает 16-бит логический маркер, или дескриптор,

который сохраняется прикладной программой и используется для задания файла в последующих операциях.

В тех случаях, когда вы применяете функции типа дескриптора, операционная система сохраняет структуры данных, которые содержат учетную информацию о файле внутри выделяемого файлу пространства памяти, однако прикладной программе эти структуры недоступны. Функции типа дескриптора полностью поддерживают иерархическую файловую структуру, обеспечивая возможность программисту создавать, открывать, закрывать и уничтожать файлы в любом подкаталоге любого диска, читать или делать записи любого размера по любому относительному адресу (в байтах) внутри файла.

Несмотря на то что в этой главе вначале обсуждаются функции типа FCB, что обусловлено историческими причинами, новые прикладные программы, работающие в среде MS-DOS, должны писаться с применением более мощных функций типа дескриптора. Следует избегать применения функций типа FCB в новых программах, если только не требуется обеспечить совместимости с системой MS-DOS версии 1.

ПРИМЕНЕНИЕ ФУНКЦИЙ ТИПА FCB

Понимание сути структуры блока управления файлами – ключ к успешной работе с FCB-функциями обращения к файлам и записям. Блок управления данными (FCB) – это 37-байт структура данных, которая расположена в пространстве памяти, выделяемой прикладной программе; FCB делится на несколько полей (рис. 8.1). Обычно программа инициализирует FCB кодом дисковода, именем файла и его расширением (для чего удобно использовать прерывание Int 21H с функцией 29H, обеспечивающей выделение и перенос составляющих имени файла), а затем передает адрес блока управления файлами системе MS-DOS с тем, чтобы открыть или создать файл. В случае успешного выполнения операций по открытию или созданию файла

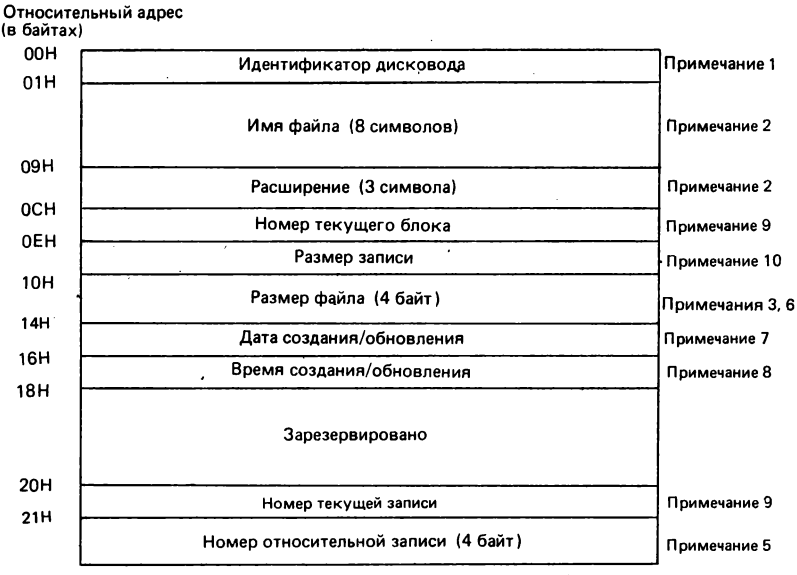


Рис. 8.1. Типичный блок управления файлом. Общая длина 37 байт (25H байт). (См. примечания к рис. 8.1 и 8.3)

MS-DOS заполняет определенные поля блока управления файлами информацией из записи каталога диска, описывающей файл. Сюда включаются точный размер файла (в байтах), а также дата и время создания или последней модификации файла. Кроме того, система MS-DOS размещает некоторые данные в области, зарезервированной в блоке управления файлами, однако эта область используется операционной системой для своих собственных целей, причем по-разному в разных версиях MS-DOS. Прикладные программы никогда не должны изменять содержимое этой области.

В целях совместимости с CP/M система MS-DOS автоматически устанавливает поле размера записи в FCB равным 128 байт. Если этот размер, устанавливаемый по умолчанию, неудобен, то программа должна после операции открытия или создания поместить в поле размера записи требуемый размер (в байтах). Позже, когда программе понадобится записать или прочитать записи файла, программа должна передать MS-DOS адрес FCB; MS-DOS в свою очередь передаст в обновляемый FCB информацию о текущем положении указателя файла и размере файла. Данные всегда считываются или пишутся из текущей области пересылки диска (DTA), адрес которой устанавливается с помощью прерывания Int 21H с функцией 1AH. Если в прикладной программе желательно реализовать режим прямого доступа к записям, то номер записи следует установить в FCB до очередного обращения к функции. В случае же последовательного доступа к записям MS-DOS сама следит за обновлением полей FCB, и никакого специального вмешательства со стороны прикладной программы не требуется.

Обычно функции MS-DOS, использующие блоки управления файлами, получают полный адрес FCB через регистр DS:DX и возвращают код завершения через регистр AL (рис. 8.2). При выполнении файловых операций (открыть, закрыть, создать и уничтожить) этот код завершения равен нулю в случае успешного выполнения функции и 0FFH (255), если функция не выполнена. Для FCB-функций записи и чтения возвращаемый через регистр AL код успешного выполнения операции по-прежнему равен нулю, однако имеется несколько кодов фатальных ошибок. В системе MS-DOS версии 3.0 и старше более подробную информацию об ошибке можно получить, вызвав прерывание Int 21H с функцией 59H (Получить расширенную информацию об ошибках) после неудачного выполнения функции типа FCB.

При загрузке программы под управлением MS-DOS операционная система формирует в префиксе программного сегмента два блока управления файлами, расположенные по относительным адресам 005CH и 006CH. Их часто называют стандартными

```
                                ; Выделение и перенос имени файла
                                ; ранее выполнен в "my_fcb"
        mov   dx,seg my_fcb     ; DS:DX = адрес блока управления
        mov   ds,dx             ; файлами
        mov   dx,offset my_fcb
        mov   ah,0fh            ; Функция 0fh = открыть
        int   21h
        or    al,al             ; Операция открыть выполнена
                                ; успешно ?
        jnz   error             ; Нет,переход на
                                ; подпрограмму error
        .
        .
        .
my_fcb  db    37 dup (0)        ; Блок управления файлами
```

Рис. 8.2. Типичная файловая FCB-операция. Этот фрагмент программы пытается открыть файл, имя которого ранее перенесено в FCB с именем my_fcb.

блоками управления файлами; они включаются для обеспечения совместимости снизу вверх с системой CP/M. Система MS-DOS выделяет первые два параметра командной строки, которая возбуждает программу (исключая любые директивы перенаправления), и переносит их в стандартные блоки управления, предполагая, что параметры могут быть спецификаторами файлов. Прикладная программа должна определить, действительно ли они представляют собой имена файлов или нет. Кроме того, так как стандартные FCB перекрываются и неудобно расположены (особенно это касается программ типа .EXE), то, как правило, перед использованием их необходимо скопировать в другое место (см. гл. 3).

Отметим, что структуры блоков управления файлами в системах CP/M и MS-DOS неидентичны. Однако различия касаются главным образом резервированных областей FCB, которые в любом случае не должны использоваться прикладными программами, поэтому корректные программы для CP/M должны относительно легко переноситься в среду MS-DOS. Но, по-видимому, таких программ мало. Многие приемы, которые использовались хитроумными CP/M-программистами для увеличения производительности или обхода ограничений этой операционной системы, могут вызывать серьезные проблемы при работе в среде MS-DOS, особенно сетевых конфигурациях. Во всяком случае гораздо большей производительности можно добиться путем полной переработки прикладных программ, ориентированных на CP/M, поскольку это дает возможность использовать мощные средства MS-DOS.

Для создания или обращения к файлам, имеющим специальные атрибуты (такие, как скрытый или доступный только для чтения), а также при работе с метками тома, подкаталогами можно использовать специальный вариант FCB, который называется

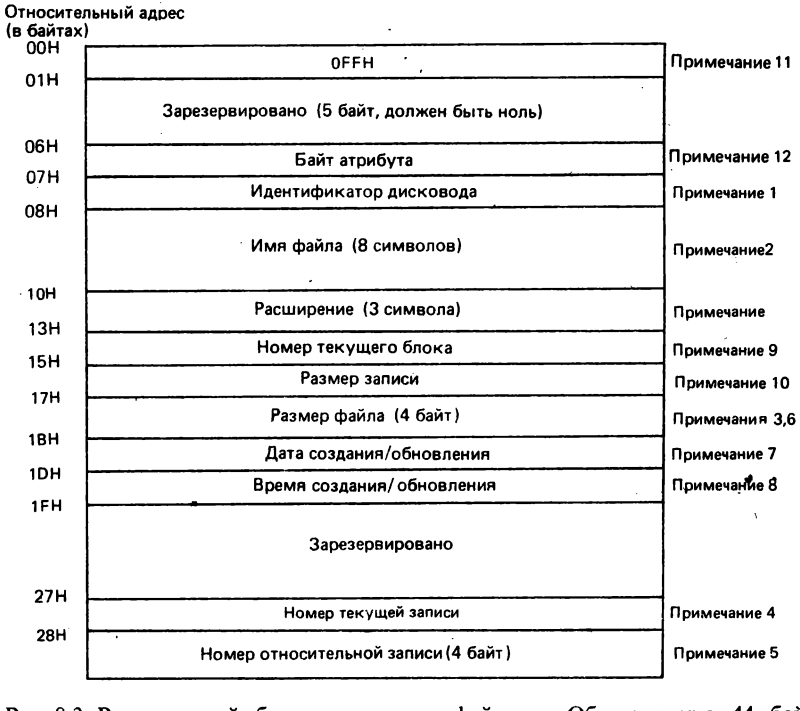


Рис. 8.3. Расширенный блок управления файлами. Общая длина 44 байт (2CH байт). (См. примечания к рис. 8.1 и 8.3.)

расширенным блоком управления файлами. Расширенный FCB имеет 7-байт заголовок, за которым следует 37-байт структура обычного блока управления файлами (рис. 8.3). В первом байте содержится число 0FFH, которое не совпадает с каким-либо допустимым кодом дисковода, что указывает MS-DOS на использование расширенного FCB. Следующие 5 байт зарезервированы и не используются в современных версиях MS-DOS. В седьмом байте содержится атрибут файла, к которому происходит обращение. (Байты атрибутов подробно обсуждаются в гл. 9.) Любая функция MS-DOS, использующая обычный блок управления файлами, может также применять расширенный FCB.

Функции типа FCB для управления файлами и записями можно разделить на группы по следующим основным признакам:

Функция	**Описание**
Обычно используемые функции типа FCB для операций с файлами	
0FH	Открыть файл
10H	Закрыть файл
16H	Создать файл
Обычно используемые функции типа FCB для операций с записями	
14H	Выполнить последовательное чтение
15H	Выполнить последовательную запись
21H	Выполнить чтение в режиме прямого доступа
22H	Выполнить запись в режиме прямого доступа
27H	Выполнить чтение блока в режиме прямого доступа
28H	Выполнить запись блока в режиме прямого доступа
Другие важные FCB-операции	
1AH	Установить адрес области пересылки диска
29H	Выделить и перенести в FCB имя файла
Реже используемые функции типа FCB для операций с файлами	
13H	Уничтожить файл
17H	Переименовать файл
Реже используемые функции типа FCB для операций с записями	
23H	Получить размер файла
24H	Установить номер относительной записи

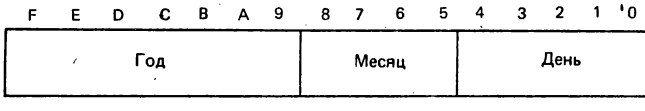
Некоторые из этих функций имеют специальные свойства. Например, операции, осуществляемые с помощью прерываний Int 21H с функцией 27H (Чтение блока прямого доступа) и функцией 28H (Запись блока прямого доступа), позволяют читать и записывать группы записей любого размера, а также автоматически обновлять запись прямого доступа (в отличие от операций, выполняемых с помощью прерывания Int 21H с функциями 21H и 22H). С помощью прерывания Int 21H с функцией 28H можно усекать файл до любого требуемого размера, а с помощью прерывания Int 21H с функцией 17H, которое применяется с расширенным FCB, можно изменять метку тома и переименовывать подкаталог.

В разд. 2 «Справочник по функциям MS-DOS» дается подробное описание всех функций типа FCB для работы с файлами и записями вместе с примерами программ на языке ассемблера. Кроме того, полезно сравнить рассмотренные группы функций типа

FCB с соответствующими группами функций типа дескриптора (см. раздел «Применение функций типа дескриптора»).

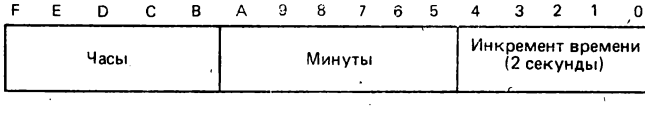
Примечание к рис. 8.1 и 8.3

1. Идентификатором дисковода является значение двоичного числа: 00 = устанавливаемый по умолчанию дисковод, 01 = дисковод A:, 02 = дисковод B: и т. д. Если в прикладной программе указан дисковод с кодом 0 (устанавливаемый по умолчанию дисковод), то MS-DOS после успешного выполнения операции открытия или создания заполняет идентификатор дисковода значением кода фактического текущего диска.
2. Имена файлов и их расширения должны выравниваться по левому краю и дополняться пробелами.
3. Размер файла, дата и время его создания, а также зарезервированные поля не должны изменяться прикладными программами.
4. Все словные поля хранятся с младшим байтом по меньшему адресу.
5. Поле относительной записи обрабатывается как 4-байт, если размер записи меньше 64 байт, в противном случае используются только первые 3 байт поля.
6. Поле размера файла имеет тот же формат, что и в каталоге, с младшим словом по меньшему адресу.
7. Поле даты устанавливается, как в каталоге. Представленное в виде 16-бит слова (как содержимое регистра) поле разбивается следующим образом:



Биты	Значение
00H–04H	День (1–31)
05H–08H	Месяц (1–12)
09H–0FH	Год (относительно 1980)

8. Поле времени устанавливается так же, как в каталоге. Представленное в виде 16-бит слова (как содержимое регистра) поле разбивается следующим образом:



Биты	Значение
00H–04H	2-секундное приращение (0–29)
05H–0AH	Минуты (0–59)
0BH–0FH	Часы (0–23)

9. Номера текущего блока и текущей записи используются совместно в операциях последовательного чтения и записи. Так имитируется поведение системы CP/M.
10. Прерывание Int 21H с функцией 0FH (открыть) и функцией 16H (создать) в целях обеспечения совместимости с системой CP/M устанавливает поле размера записи равным 128 байт. Если используется другой размер записи, то после выполнения операции открытия или создания необходимо занести его в поле.
11. Число 0FFH (255) в первом байте структуры означает, что это расширенный

блок управления файлами. Можно применять расширенный FCB со всеми функциями, использующими обычный FCB (см. также примечание 12).

12. Байт атрибутов в расширенном FCB дает возможность обращаться к файлам, имеющим специальные характеристики: скрытый, системный или доступный только для чтения. Можно также использовать расширенный блок управления файлами для чтения меток тома и содержимого специальных файлов подкаталогов.

Структура программы работы с файлами с помощью функций типа FCB

Ниже приведена типичная программа обращения к файлам с помощью функций типа FCB или традиционных функций (рис. 8.4):

1. Обнулить будущий блок управления файлами.
2. Получить имя файла от пользователя, из стандартного FCB или из хвоста командной строки в PSP.

```
recsize   equ   1024                ; Размер записи файла
          .
          .
          .
          mov   ah,29h              ; Выделение и перенос имени вход-
                                    ; ного файла
          mov   al,1                ; Удалить начальные пробелы
          mov   si,offset fname1    ; Адрес имени файла
          mov   di,offset fcb1      ; Адрес FCB
          int   21h
          or    al,al               ; Перейти, если имя
          jnz   name_err            ; задано неверно
          .
          .
          .
          mov   ah,29h              ; Выделение и перенос имени выход-
                                    ; ного файла
          mov   al,1                ; Удалить начальные пробелы
          mov   di,offset fname2    ; Адрес имени файла
          mov   di,offset fcb2      ; Адрес FCB
          int   21h
          or    al,al               ; Перейти, если имя
          jnz   name_err            ; неверно
          .
          .
          mov   ah,0fh              ; Открыть входной файл
          mov   dx,offset fcb1
          int   21h
          or    al,al               ; Операция открыть выполнена
                                    ; успешно ?
          jnz   no_file             ; Нет, перейти

          .
          .
          .
          mov   ah,16h              ; Создать и открыть
          mov   dx,offset fcb2      ; выходной файл
          int   21h
          or    al,al               ; Создание выполнено успешно ?
```

Рис. 8.4. Структура программы на языке ассемблера, осуществляющей операции ввода-вывода файлов и записей с помощью функций типа FCB.

```
          jnz   disk_full              ; Нет, перейти
          .
          .
          .                            ; Установить размер записи
          mov   word ptr fcb1+0eh,recsize
          mov   word ptr fcb2+0eh,recsize
          .
          .
          .
          mov   ah,1ah                 ; Установить адрес области пересылки
          mov   dx,offset buffer       ; диска для операций чтения
          int   21h                    ; и записи
          .
next:     .                            ; Выполнить следующую запись
          .
          mov   ah,14h                 ; Последовательное чтение
          mov   dx,offset fcb1         ; входного файла
          int   21h
          cmp   al,01                  ; Проверить достижение конца файла
          je    file_end               ; Перейти, если конец файла
                                       ; достигнут
          cmp   al,03
          je    file_end               ; Перейти, если конец файла
                                       ; достигнут
          or    al,al                  ; Произошла другая ошибка чтения ?
          jnz   bad_read               ; Перейти, если ошибка чтения
          .
          .
          .
          mov   ah,15h                 ; Последовательная запись
          mov   dx,offset fcb2         ; выходного файла
          int   21h
          or    al,al                  ; Запись завершена успешно ?
          jnz   bad_write              ; Перейти, если запись не выполнена
          .
          .
          .
          jmp   next                   ; Выполнить следующую запись
          .
file_end: .                            ; Достигнут конец операции ввода
          .
          mov   ah,10h                 ; Закрыть входной файл
          mov   dx,offset fcb2
          int   21h
          .
          .
          .
          mov   ah,10h                 ; Закрыть выходной файл
          mov   dx,offset fcb2
          int   21h
          .
          .
          .
          mov   ax,4c00h               ; Выйти с кодом
          int   21h                    ; завершения 0
          .
          .
          .
```

Рис. 8.4.

```
fname1      db    'OLDFILE.DAT'.0        ; Имя входного файла
fname2      db    'NEWFILE.DAT'.0        ; Имя выходного файла
fcb1        db    37 dup (0)             ; FCB входного файла
fcb2        db    37 dup (0)             ; FCB выходного файла
buffer      db    recsize dup(?)         ; Буфер файла ввода-вывода
```

Рис. 8.4.

Относительный адрес (в байтах)	FCB до открытия	Содержимое FCB	FCB после открытия
00H	00	Дисковод	03
01H	4D	Имя файла	4D
02H	59		59
03H	46		46
04H	49		49
05H	4C		4C
06H	45		45
07H	20		20
08H	20		20
09H	44	Расширение	44
0AH	41		41
0BH	54		54
0CH	00	Текущий блок	00
0DH	00		00
0EH	00	Размер записи	80
0FH	00		00
10H	00	Размер файла	80
11H	00		3D
12H	00		00
13H	00		00
14H	00	Дата создания файла	43
15H	00		0B
16H	00	Время создания файла	A1
17H	00		52
18H	00	Зарезервировано	03
19H	00		02
1AH	00		42
1BH	00		73
1CH	00		00
1DH	00		01
1EH	00		35
1FH	00		0F
20H	00	Текущая запись	00
21H	00	Номер относительной записи	00
22H	00		00
23H	00		00
24H	00		00

Рис. 8.5. Содержимое типичного блока управления файлами до и после успешно выполненной операции открытия (с помощью прерывания Int 21H с функцией OFH).

3. Если имя файла не было получено из стандартного FCB, то с помощью прерывания Int 21H с функцией 29H перенести имя файла в новый блок управления файлами.

4. С помощью прерывания Int 21H с функцией 0FH открыть файл, или, если осуществляется запись новых данных, создать файл, или усечь уже существующий файл с тем же именем до нулевой длины (с помощью прерывания Int 21H с функцией 16H).

5. Установить в FCB поле размера записи, если только не используется устанавливаемый по умолчанию размер записи. Вспомним, что выполнить это действие необходимо после успешного завершения операции открытия или создания (рис. 8.5).

6. Установить в FCB поле относительной записи, если осуществляется операция ввода-вывода записи в режиме прямого доступа.

7. С помощью прерывания Int 21H с функцией 1AH установить адрес области пересылки диска, если адрес буфера изменялся с момента последнего обращения к этой функции. Если прикладная программа не осуществляет установку DTA, то по умолчанию адрес DTA устанавливается в префиксе в поле с относительным адресом 0080H.

8. С помощью прерывания Int 21H сделать запрос на необходимую операцию чтения или записи (с функцией 14H – последовательное чтение, 15H – последовательная запись, 21H – чтение в режиме прямого доступа, 22H – запись в режиме прямого доступа, 27H – чтение блока в режиме прямого доступа, 28H – запись блока в режиме прямого доступа).

9. Если программа не завершила обработку файла, то вернуться к пункту 6, в противном случае выполнить (с помощью прерывания Int 21H с функцией 10H) операцию закрытия файла. Если файл только читался, то в ранних версиях MS-DOS операцию закрытия можно пропустить. Однако такое сокращение может вызвать проблемы при работе в среде MS-DOS версии 3.0 и старше, особенно когда доступ к файлам осуществляется по сети.

Памятка

Рассмотрим все за и против использования в программах функций типа FCB для осуществления доступа к файлам и записям.

Преимущества:

- При работе с функциями типа FCB в среде MS-DOS версии 1 и 2 число файлов, которые можно открыть одновременно, ограничено. (В среде MS-DOS версии 3.0 и старше это не так, особенно если активизировано сетевое обеспечение.)
- Методы доступа к файлам с помощью функций типа FCB знакомы программистам, работавшим с системой CP/M, а корректные прикладные программы для CP/M требуют всего лишь небольших изменений в логике с тем, чтобы выполнить их в среде MS-DOS.
- Система MS-DOS передает в блок управления файлами размер, время и дату создания файла после его открытия. Вызывающая программа может проанализировать эту информацию.

Недостатки:

- Блоки управления файлами занимают пространство памяти, выделяемое прикладной программе.
- Блоки управления файлами не обеспечивают поддержку иерархической файловой структуры (нет доступа к файлам за пределами текущего каталога).
- Блоки управления файлами не обеспечивают поддержку запирания/разделения файла, а также запирания записей в среде сетевой конфигурации.
- В дополнение к самим операциям чтения или записи ввод-вывод файла с использованием FCB требует обработки блока управления файлами с целью установки

размера и номера записи плюс еще предварительный вызов отдельной функции MS-DOS для установки адреса DTA.

- Ввод-вывод записей в режиме прямого доступа для файла с записями переменной длины с помощью функций типа FCB громоздок и неудобен.
- Для доступа к файлам или создания файлов со специальными атрибутами, такими, как «скрытый», «только для чтения» или «системный», необходимо использовать расширенные блоки управления файлами, несовместимые с системой CP/M.
- Функции типа FCB для работы с файлами характеризуются слабыми возможностями диагностики ошибок. Несколько лучше обстоит дело в системе MS-DOS версии 3, где программа после невыполненной FCB-операции может для получения дополнительной информации обратиться к прерыванию Int 21H с функцией 59H (Получить расширенную информацию об ошибке).
- Фирма Microsoft не рекомендует использовать FCB. Наличие в программе FCB затруднит перенесение такой программы в среду MS OS/2, которая совсем не поддерживает FCB-операции при работе в защищенном режиме.

ПРИМЕНЕНИЕ ФУНКЦИЙ ТИПА ДЕСКРИПТОРА

Функции управления файлами и записями типа дескриптора обращаются к файлам аналогично тому, как это делается в среде операционных систем UNIX/XENIX. Файлы указываются с помощью ASCIIZ-строки (строки, состоящей из символов в коде ASCII и заканчивающейся нулевым байтом), которая содержит обозначение дисковода, путь, имя файла и расширение. Например, описание файла

C: SYSTEM COMMAND.COM

кодируется в памяти следующей последовательностью байт:

43 3A 5C 53 59 53 54 45 4D 5C 43 4F 4D 4D 41 4E 44 2E 43 4F 4D 00

Если в программе нужно открыть или создать файл, она пересылает MS-DOS адрес ASCIIZ-строки, задающей этот файл, через регистры DS:DX (рис. 8.6). В случае успешного выполнения этой операции MS-DOS возвращает программе через регистр

```
                mov   ah,3dh                ; Функция 3dh = открыть
                mov   al,2                  ; Режим 2 = чтение / запись
                mov   dx,seg filename       ; Адрес ASCIIZ - строки
                mov   ds,dxq                ; спецификации файла
                mov   dx,seg filename
                int   21h                   ; Системный запрос на открытие
                jc    error                 ; Перейти, если операция
                                            ; окрыть не выполнена
                mov   handle,ax             ; Сохранить дескриптор файла
                .
                .
                .
filename        db    'C:\MYDIR\MYFILE.DAT'.0 ; Имя файла
handle          dw    0                     ; Дескриптор файла
```

Рис. 8.6. Типичная операция с файлами с помощью функций типа дескриптора.

В этом фрагменте программы показана попытка открыть файл, образованный строкой ASCIIZ, адрес которой передается системе MS-DOS через регистры DS:DX.

AX 16-бит дескриптор. Программа должна сохранить этот дескриптор, чтобы иметь возможность в дальнейшем на него ссылаться.

Когда в программе запрашиваются последующие файловые операции, то перед обращением к MS-DOS дескриптор помещается в регистр BX. Все функции типа дескриптора возвращают флаг переноса слова состояния процессора сброшенным, если операция выполнена успешно, и установленным, если операция не выполнена. В последнем случае в регистре AX содержится код, описывающий причину невыполнения.

Система MS-DOS ограничивает число дескрипторов, которые могут быть одновременно активны (т.е. число файлов и устройств, которые можно одновременно открыть, если использовать функции типа дескриптора), двумя способами:

- Максимальное число одновременно открытых файлов в системе для всех активных процессов задается в файле CONFIG.SYS записью FILES = nn. Тем самым определяется число записей, которые можно использовать в системной таблице файлов. В системе MS-DOS версии 3 это значение по умолчанию равно 8, а максимальное значение – 255. После загрузки и запуска MS-DOS нельзя расширить эту таблицу с тем, чтобы увеличить общее число файлов, которые могут быть одновременно открыты. Необходимо использовать текстовый редактор, чтобы модифицировать файл CONFIG.SYS, а затем выполнить повторный запуск системы.
- Максимальное число одновременно открытых файлов для одного процесса равно 20 при условии, что в системной таблице файлов имеется достаточное количество записей. В момент загрузки программы система MS-DOS сразу же 5 из 20 возможных дескрипторов назначает своим стандартным устройствам. Всякий раз, когда данный процесс обращается к операции открытия или создания, MS-DOS выделяет дескриптор, один из 20 доступных данному процессу, пока либо не исчерпаются все дескрипторы, либо не заполнится вся системная таблица файлов. В системе MS-DOS версии 3.3 и старше можно расширить установленное ограничение (20 дескрипторов) с помощью прерывания Int 21H с функцией 67H (Установить число дескрипторов).

Функции управления файлами и записями типа дескриптора можно в целях изучения разделить на группы по следующим основным признакам:

Функция	Описание
	Обычно используемые функции типа дескриптора для операций с файлами
3CH	Создать файл (запрашивает ASCIIZ-строку)
3DH	Открыть файл (запрашивает ASCIIZ-строку)
3EH	Закрыть файл
	Обычно используемые функции типа дескриптора для операций с записями
42H	Установить указатель файла (также применяется для нахождения размера файла)
3FH	Прочитать из файла
40H	Записать в файл
	Реже используемые функции типа дескриптор
41H	Удалить файл
43H	Получить или установить атрибуты файла
44H	Управление вводом-выводом (IOCTL)
45H	Дублировать дескриптор
46H	Перенаправить дескриптор
56H	Переименовать файл
57H	Получить или установить дату и время создания файла

5AH	Создать временный файл (версии 3.0 и старше)
5BH	Создать новый файл (не выполняется, если файл уже существует; версии 3.0 и старше)
67H	Установить число дескрипторов
68H	Сбросить в файл (версии 3.3 и старше)
6CH	Расширенное открытие файла (версия 4)

Сравнивая группы функций типа дескриптора из представленной таблицы с рассмотренными ранее группами функций типа FCB, отметим их значительное функциональное перекрытие. В разд. 2 «Справочник по функциям MS-DOS» дается подробное описание всех функций типа дескриптора вместе с примерами программ на языке ассемблера.

Структура программы работы с файлами с помощью функций типа дескриптора

Ниже приведена типичная программа обращения к файлам с помощью функций типа дескриптора (рис. 8.7):

1. Получить имя файла от пользователя посредством буферизованного ввода (с помощью прерывания Int 21H с функцией 0AH) или из хвоста командной строки, заносимой системой MS-DOS в PSP.

```
recsize     equ     1024                ; Размер записи файла
            .
            .
            .
            mov     ah,3dh              ; Открыть выходной файл
            mov     al,0                ; Режим = только для чтения
            mov     dx,offset fname1    ; Имя входного файла
            int     21h
            jc      no_file             ; Перейти, если файл отсутствует
            mov     handle,ax           ; Сохранить дескриптор файла
            .
            .
            .
            mov     ah,3ch              ; Создать выходной файл
            mov     cx,0                ; Атрибут = обычный
            mov     dx,offset fname2    ; Имя выходного файла
            int     21h
            jc      disk_full           ; Перейти, если операция создать
                                        ; не выполнена
            mov     handle2,ax          ; Сохранить дескриптор файла
            .
next:       .                           ; Выполнить следующую запись
            .
            mov     ah,3fh              ; Последовательное чтение
            mov     bx,handle1          ; входного файла
            mov     cx,recsize
            mov     dx,offset buffer
            int     21h
            jc      bad_read            ; Перейти, если ошибка чтения
            or      ax,ax               ; Проверить переданные байты
```

Рис. 8.7. Структура программы на языке ассемблера, которая осуществляет последовательное выполнение операций ввода файла и записи результатов в выходной файл с помощью функций типа дескриптора.

Программа предполагает, что регистры DS и ES уже установлены, так что указывают на сегмент, содержащий буферы и имена файлов.

```
            jz    file_end              ; Перейти,если достигнут конец файла
            .
            .
            .
            mov   ah,40h                ; Последовательная запись в
            mov   bx,handle2            ; выходной файл
            mov   cx,recsize
            mov   dx,offset buffer
            int   21h
            jc    bad_write             ; Перейти, если ошибка записи
            cmp   ax,recsize            ; Запись выполнена полностью ?
            jne   disk_full             ; Перейти, если диск полон
            .
            .
            .
            jmp   next                  ; Выполнить следующую запись
            .
file_end:   .                           ; Завершение конца операции ввода
            .
            mov   ah,3eh                ; Закрыть выходной файл
            mov   bx,handle1
            int   21h
            .
            .
            .
            mov   ah,3eh                ; Закрыть выходной файл
            mov   bx,handle2
            int   21h
            .
            .
            .
            mov   ax,4c00h              ; Закончить с кодом
            int   21h                   ; завершения 0
            .
            .
            .
fname1      db    'OLDFILE.DAT',0       ; Имя входного файла
fname2      db    'NEWFILE.DAT',0       ; Имя выходного файла
handle1     dw    0                     ; Дескриптор входного файла
handle2     dw    0                     ; Дескриптор выходного файла
buffer      db    recsize dup(?)        ; Буфер файла ввода / вывода
```

Рис. 8.7.

2. Поместить нуль в конец спецификации файла с целью создания ASCIIZ-строк.

3. Открыть файл с помощью прерывания Int 21H с функцией 3DH в режиме 2 (доступ чтения-записи) или создать файл с помощью прерывания Int 21H с функцией 3CH. (Не забудьте записать 0 в регистр CX, чтобы случайно не создать файл со специальными атрибутами.) Сохранить возвращенный дескриптор.

4. Установить указатель файла с помощью прерывания Int 21H с функцией 42H. Можно установить положение указателя файла относительно одной из трех позиций: начала файла, текущего положения указателя и конца файла. При выполнении последовательного ввода-вывода обычно можно пропустить этот шаг, так как система MS-DOS автоматически модифицирует положение указателя файла.

5. Прочитать из файла (с помощью прерывания Int 21H с функцией 3FH) или записать в файл (с помощью прерывания Int 21H с функцией 40H). Обе эти функции требуют, чтобы регистр BX содержал дескриптор файла, регистр CX – длину записи,

а регистры DS:DX указывали на передаваемые данные. Обе функции возвращают через регистр AX общее число переданных байт.

Если во время операции чтения количество прочитанных из файла байт меньше, чем было затребовано, то это означает, что достигнут конец файла. Если при выполнении операции записи количество записанных из файла байт меньше, чем затребовано, то это означает, что диск, содержащий файл, полностью заполнен. Ни одно из перечисленных условий не возвращает кода ошибки, т. е. флаг переноса не устанавливается.

6. Если программа не завершила обработку файла, то вернуться к пункту 4, в противном случае выполнить (с помощью прерывания Int 21H с функцией 3EH) операцию закрытия файла. Всякий нормальный выход из программы также закроет все активные дескрипторы.

Памятка

Подведем итог всех за и против использования в программе операций доступа к файлам и записям с помощью функций типа дескриптора. Сравните приведенный ниже перечень с ранее рассмотренными семействами функций типа FCB.

Преимущества:

- Обращения к дескриптору обеспечивают прямую поддержку операций перенаправления ввода-вывода и конвейеризации для устройств стандартного ввода и вывода функционально аналогично тому, как это делается в среде UNIX/XENIX.
- Функции типа дескриптора обеспечивают непосредственную поддержку каталогов (иерархической структуре файлов) и специальных файловых атрибутов.
- Обращения к дескриптору поддерживают разделение (запирание файла и запирание записи в сетевых конфигурациях).
- С помощью функций типа дескриптора программист может открывать каналы для символьно-ориентированных устройств и работать с ними, как с файлами.
- Обращения к дескриптору делают работу с записями в режиме прямого доступа очень простым делом. Указатель текущего файла можно установить на любой относительный адрес (в байтах) от начала файла, конца файла или текущего положения курсора. За одну операцию можно прочитать в любое место памяти запись любой длины, вплоть до полного сегмента (65535 байт).
- Функции типа дескриптора характеризуются относительно неплохой диагностикой ошибок в MS-DOS версии 2; в MS-DOS версии 3.0 и старше система диагностики получила еще большее развитие.
- Фирма Microsoft всячески поощряет применение функций типа дескриптора с тем, чтобы обеспечить совместимость снизу вверх с системой MS OS/2.

Недостатки:

- При работе с дескрипторами в среде MS-DOS версий 2.0–3.2 существует предел, равный 20, на число одновременно открытых в одной программе файлов и устройств.
- В системе функций типа дескриптора до сих пор остаются небольшие изъяны. Например, для изменения меток тома и доступа к содержимому специальных файлов, реализующих каталоги, все еще необходимо использовать расширенные блоки управления файлами.

СИСТЕМНЫЕ КОДЫ ОШИБОК

В тех случаях, когда файловые операции функций типа дескриптора не выполняются и устанавливается флаг переноса, а также когда в программе вызывается вслед за невыполненной FCB-операцией или другой операцией системного обслуживания прерывание Int 21H с функцией 59H (Получить расширенную информацию об ошибках),

в программу возвращается один из следующих кодов ошибки:

Значение	Описание
Коды ошибок системы MS-DOS версии 2	
01H	Неверный номер функции
02H	Файл не найден
03H	Путь не найден
04H	Открыто слишком много файлов
05H	Доступ запрещен
06H	Недопустимый дескриптор
07H	Уничтожены блоки управления памятью
08H	Не хватает памяти
09H	Недопустимый адрес блока памяти
0AH(10)	Недопустимый контекст
0BH(11)	Недопустимый формат
0CH(12)	Недопустимый код доступа
0DH(13)	Недопустимые данные
0EH(14)	Неизвестное устройство
0FH(15)	Недопустимый дисковод
10H(16)	Попытка удалить текущий каталог
11H(17)	Не то же устройство
12H(18)	Нет больше файлов
Отображения кодов критических ошибок	
13H(19)	Попытка записи на запрещенный диск
14H(20)	Неизвестное устройство
15H(21)	Дисковод не готов
16H(22)	Неизвестная команда
17H(23)	Ошибка данных (CRC)
18H(24)	Неверная длина структуры запроса
19H(25)	Ошибка установки позиционирования
1AH(26)	Неизвестный тип носителя
1BH(27)	Сектор не найден
1CH(28)	В принтере нет бумаги
1DH(29)	Ошибка записи
1EH(30)	Ошибка чтения
1FH(31)	Общая ошибка
Коды расширенных ошибок системы MS-DOS версии 3 и старше	
20H(32)	Нарушение разделения
21H(33)	Нарушение запирания файла
22H(34)	Недопустимая смена дискеты
23H(35)	FCB отсутствует
24H(36)	Разделяемый буфер переполнен
25H–31H(37–49)	Зарезервирован
32H(50)	Неподдерживаемый сетевой запрос
33H(51)	Удаленная машина не слушает
34H(52)	Дублирование имени в сети
35H(53)	Сетевое имя не найдено
36H(54)	Сеть занята

37H(55)	Устройство больше не существует в сети
38H(56)	Превышен предел команд NetBIOS
39H(57)	Аппаратная ошибка сетевого адаптера
3AH(58)	Неправильный отклик сети
3BH(59)	Неожиданная ошибка сети
3CH(60)	Несовместимость удаленного адаптера
3DH(61)	Очередь к принтеру заполнена
3EH(62)	Недостаточно места для файла принтера
3FH(63)	Файл принтера удален
40H(64)	Сетевое имя удалено
41H(65)	Доступ к сети невозможен
42H(66)	Неправильный тип сетевого устройства
43H(67)	Сетевое имя не найдено
44H(68)	Превышен предел сетевого имени
45H(69)	Превышен предел сетевого сеанса NetBIOS
46H(70)	Рабочая пауза
47H(71)	Сетевой запрос не принят
48H(72)	Приостановлено перенаправление принтера или диска
49H–4FH(73–79)	Зарезервированы
50H(80)	Файл уже существует
51H(81)	Зарезервированы
52H(82)	Каталог не может быть создан
53H(83)	Отказ по прерыванию Int 21H (критическая ошибка)
54H(84)	Слишком много перенаправлений
55H(85)	Двойное перенаправление
56H(86)	Неверный пароль
57H(87)	Неверный параметр
58H(88)	Ошибка записи по сети

В среде системы MS-DOS версии 3.0 и старше с помощью прерывания Int 21H с функцией 59H можно также получить дополнительную информацию об ошибке, например место, где произошла ошибка, и действия, рекомендуемые по устранению ошибки.

Обработчик прерываний критической ошибки

В гл. 5 уже обсуждалось, каким образом прикладная программа захватывает вектор обработчика прерывания Ctrl-C (с помощью прерывания Int 23H) и замещает стандартный системный обработчик прерывания с целью избежать потери управления компьютером при вводе пользователем с клавиатуры комбинации клавиш Ctrl-C и Ctrl-Break. Подобным же образом система MS-DOS предоставляет вектор обработчика прерывания критической ошибки (с помощью прерывания Int 24H), который определяет программу, вызываемую в случае возникновения невосстанавливаемой аппаратной ошибки. Стандартный системный обработчик прерывания критической ошибки – это программа, которая высвечивает сообщение, описывающее тип ошибки, а также запрос:

Abort, Retry, Ignore? (Прервать, повторить, игнорировать?)

Это сообщение появляется после следующих действий:

- Попытка открыть файл на дисководе при отсутствии дискеты или при открытом кармане

- Попытка прочитать сектор диска, содержащий ошибку CRC
- Попытка вывода на неподключенный принтер

Неприятная особенность стандартного системного обработчика прерываний критической ошибки состоит в том, что если пользователь вводит символ A для выполнения команды Abort (прервать), то выполняющаяся в этот момент прикладная программа тут же завершается, не имея возможности «выполнить уборку» и совершить изящный выход. Промежуточные файлы останутся на диске; файлы, использующие FCB, закроются неправильно, что приведет к модификации каталога; векторы прерываний будут указывать на область транзитной программы и т. д.

Для того чтобы написать устойчивую к неверным командам прикладную программу в системе MS-DOS, необходимо захватить вектор обработчика прерываний критической ошибки и настроить его на собственную программу так, чтобы ваша программа перехватывала все катастрофические аппаратные ошибки и соответственно обрабатывала их. Для того чтобы изменить вектор прерывания Int 24H корректным образом, можно использовать системное прерывание Int 21H с функцией 25H. При завершении программы система MS-DOS автоматически восстанавливает предыдущее содержимое вектора прерывания Int 24H по информации, сохраненной в префиксе сегмента программы.

Система MS-DOS обращается к обработчику прерываний по коду критической ошибки для двух основных классов ошибок: связанных с работой диска и не связанных с работой диска, и в зависимости от класса передает через регистры в обработчик прерываний разную информацию.

В случае ошибок, связанных с работой диска, система MS-DOS устанавливает регистры так, как показано ниже. (Биты 3–5 регистра AH устанавливаются только MS-DOS версии 3.1 и старше.)

Регистр	**Бит (ы)**	**Описание**
AH	7	0 означает ошибку диска
	6	Зарезервирован
	5	0 = запрещен ответ «игнорировать» 1 = разрешен ответ «игнорировать»
	4	0 = запрещен ответ «повторить» 1 = разрешен ответ «повторить»
	3	0 = запрещен ответ «продолжить» 1 = разрешен ответ «продолжить»
	1–2	Область, где зафиксирована ошибка диска 00 = область MS-DOS 01 = таблица размещения файлов 10 = корневой каталог 11 = область файлов
	0	0 = операция чтения 1 = операция записи
AL	0–7	Код дисковода (0 = A, 1 = B и т. д.)
DI	0–7	Код ошибки драйвера
	8–15	Не используются
BP:SI		Сегмент: относительный адрес заголовка драйвера устройства

В случае ошибок, не связанных с работой диска, прерывание обусловлено либо ошибкой символьно-ориентированного устройства, либо искажением образа памяти в таблице размещения файлов. В этом случае MS-DOS устанавливает регистры следующим образом:

Регистр	Бит (ы)	Описание
AH	7	1, признак ошибки, не связанной с работой диска
DI	0–7	Код ошибки драйвера
	8–15	Не используются
BP:SI		Сегмент:относительный_адрес заголовка драйвера символьно-ориентированного устройства

Для того чтобы определить, связана ли критическая ошибка с работой символьно-ориентированного устройства, используют адрес, содержащийся в регистрах BP:SI, с помощью которого проверяют слово с относительным адресом 0004H в заголовке драйвера, «подозреваемого» символьно-ориентированного устройства, содержащее атрибут этого устройства. Если бит 15 установлен, то ошибка действительно обусловлена символьно-ориентированным устройством, а в программе можно проанализировать поле имени заголовка драйвера и определить устройство.

К моменту входа в обработчик критических ошибок система MS-DOS уже запретила прерывания и установила стек так, как изображено на рис. 8.8. Обработчик прерываний критических ошибок не может применять никаких системных средств, за исключением прерывания Int 21H с функциями 01H–0CH (Традиционный символьно-ориентированный ввод-вывод), прерывания Int 21H с функцией 30H (Получить версию MS-DOS) и прерывания Int 21H с функцией 59H (Получить расширенную информацию

Рис. 8.8. Стек при входе в обработчик прерывания по критической ошибке.

об ошибке). Эти функции используют специальный стек, так что контекст исходной функции (вызвавшей критическую ошибку) не будет потерян.

Обработчик критической ошибки должен возвращать управление в систему MS-DOS с помощью операции IRET, передавая при этом через регистр AL один из следующих кодов действия:

Код	Описание
0	Игнорировать ошибку (MS-DOS действует, как будто обращение к исходной функции было успешным)
1	Повторить операцию
2	Завершить процесс, в котором неожиданно произошла ошибка
3	Продолжить выполнение, пропустив функцию, дающую ошибку (код ошибки возвращается в запрашивающий процесс). Только для версий 3.1 и старше

Обработчик критической ошибки должен сохранять все отдельные регистры; он не может изменять заголовок драйвера устройства, на который указывают регистры BP:SI. На рис. 8.9 приведен пример структуры программы обработчика прерываний по критической ошибке.

```
                                ; Текст запросов, используемых
                                ; обработчиком критической ошибки
prompt  db      cr,lf,'Произошла критическая ошибка: '
        db      'Завершить, Повторить, Игнорировать, Пропустить? $'

keys    db      'aArRiIfF'      ; Доступные пользователю клавиши отклика
keys_len equ $-keys             ; ( возможны оба положения регистра )

codes   db      2,2,1,1,0,0,3,3 ; Возвращаемые в ядро MS-DOS коды  в
                                ; соответствии с клавишами отклика

;
; Эта программа выполняется при инициализации программы пользователя
; с целью установки нового обработчика критической ошибки.
;
        .
        .
        .
        push    ds              ; Сохранить наш сегмент данных

        mov     dx,seg int24    ; DS:DX = адрес обработчика
        mov     ds,dx
        mov     dx,offset int24
        mov     ax,2524h        ; Функция 25h = установить вектор
        int     21h             ; Переход в MS-DOS

        pop     ds              ; Восстановить сегмент данных
        .
        .
        .
;
; Здесь происходит замещение обработчика критической ошибки. Пользователю
; посылается запрос на выполнение одной из операций: Завершить,
; Повторить, Игнорировать, Пропустить,   и выполняется передача
```

Рис. 8.9. Пример структуры альтернативного обработчика прерывания по критической ошибке.

```
; кода отклика в ядро MS-DOS.
;
int24   proc    far             ; Вход из ядра MS-DOS


        push    bx              ; Сохранить регистры
        push    cx
        push    dx
        push    si
        push    di
        push    bp
        push    ds
        push    es
int24a: mov     ax,seg prompt   ; Вывести на экран запрос  пользователю
        mov     ds,ax           ; с помощью функции 9 ( печать строки
        mov     es,ax           ; с символом $ в конце )
        mov     dx,offset prompt
        mov     ah,9
        int     21h

        mov     ah,1            ; Получить отклик пользователя
        int     21h             ; Функция 1 = чтение одного символа

        mov     di,offset keys  ; Поиск кода клавиши отклика
        mov     cx,keys_len
        cld
        repne scasb
        jnz     int24a          ; Повторить запрос в случае
                                ; неверного отклика

                                ; В соответствии с откликом пользователя
                                ; установить AL = код действия MS-DOS:
                                ; 0 = игнорировать, 1 = повторить,
                                ; 2 = завершить, 3 = пропустить
        mov     al,[di+keys_len-1]

        pop     es              ; Восстановить регистры
        pop     ds
        pop     bp
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        iret                    ; Выход из обработчика критической ошибки

int24   endp
```

Рис. 8.9.

ПРИМЕРЫ ПРОГРАММ DUMP.ASM И DUMP.C

Программы DUMP.ASM (рис. 8.10) и DUMP.C (рис. 8.11) – это примеры использования функций типа дескриптора для обращения к файлам и записям в программах на языке ассемблера и Си. Вариант программы на языке ассемблера, в частности, демонстрирует особенности корректных утилит системы MS-DOS:

- Программа проверяет версию системы MS-DOS с тем, чтобы гарантировать реальную доступность всех функций, которые будут в ней использоваться.

```
        name    dump
        page    55,132
        title   DUMP--Вывод на экран содержимого файла

;
;  DUMP-- Вывод на экран содержимого файла в кодах ASCII и шестнадцатиричном
;         представлении
;
;  Build:  C>MASM DUMP;
;          C>LINK DUMP;
;
;  Usage:  C>DUMP unit:\path\filename.exe [ >device ]
;
;  Авторские права (C) 1988 Рей Данкан
;

cr      equ     0dh             ; Код ASCII возврата каретки
lf      equ     0ah             ; Код ASCII перевода строки
tab     equ     09h             ; Код ASCII табуляции
blank   equ     20h             ; Код ASCII пробела

cmd     equ     80h             ; Буфер для хвоста команды

blksize equ     16              ; Размер записи входного файла

stdin   equ     0               ; Дескриптор стандартного ввода
stdout  equ     1               ; Дескриптор стандартного вывода
stderr  equ     2               ; Дескриптор стандартной ошибки

_TEXT   segment word public 'CODE'

        assume  cs:_TEXT,ds:_DATA,es:_DATA,ss:STACK

dump    proc    far             ; Точка входа из MS-DOS

        push    вы              ; Сохранение DS:0000 для завершающего
        xor     ax,ax           ; возврата в MS-DOS в тех случаях, когда
        push    ax              ; нельзя использовать функцию 4ch

        mov     ax,_DATA        ; Сделаем наш сегмент данных
        mov     ds,ax           ; адресуемый через регистр DS

                                ; Проверка текущей версии MS-DOS
        mov     ax,3000h        ; Функция 30h = получить номер версии
        int     21h             ; Переход в MS-DOS
        cmp     al,2            ; Номер данной версии 2 или старше ?
        jae     dump1           ; Да, продолжить

                                ; Вывод на экран сообщения об ошибке
                                ; и завершение работы,
                                ; если версия MS-DOS 1.x
        mov     dx,offset msg3  ; DS:DX = адрес сообщения
        mov     ah,9            ; Функция 9 = печать строки
        int     21h             ; Переход в MS-DOS
        ret                     ; Затем выйти старым способом

dump1:                          ; Проверка наличия имени файла
        mov     bx,offset cmd   ; ES:BX = хвост команды
```

Рис. 8.10. Версия программы DUMP.ASM на языке ассемблера.

```
        call    argc            ; Подсчет числа аргументов команды
        cmp     ax,2            ; Имеется 2 аргумента ?
        je      dump2           ; Да, продолжить

                                ; Вывод на экран сообщения об ошибке
                                ; и завершение работы в случае
                                ; отсутствия имени файла
        mov     dx,offset msg2  ; DS:DX = адрес сообщения
        mov     cx,msg2_len     ; CX = длина сообщения
        jmp     dump9           ; Начать вывод сообщения

dump2:                          ; Получение адреса имени файла
        mov     ax,1            ; AX = номер аргумента,
                                ; как и ранее, ES:BX = хвост команды
        call    argv            ; Возвращает в ES:BX = адрес
                                ; и AX = длину

        mov     di,offset fname ; Скопировать имя файла в буфер
        mov     cx,ax           ; CX = длина
dump3:  mov     al,es:[bx]      ; Скопировать один байт
        mov     [di],al
        inc     bx              ; Инкремент указателя строки
        inc     di
        loop    dump3           ; Выполнить цикл до исчерпания строки
        mov     byte ptr [di],0 ; Добавить признак конца -нулевой байт

        mov     ax,ds           ; Сделаем наш сегмент данных
        mov     es,ax           ; адресуемый также через ES

                                ; Открытие файла
        mov     ax,3d00h        ; Функция 3dh = открыть файл
                                ; Режим 0 = только для чтения
        mov     dx,offset fname ; DS:DX = имя файла
        int     21h             ; Переход в MS-DOS
        jnc     dump4           ; Перейти, если операция открыть
                                ; прошла успешно

                                ; Вывод сообщения об ошибке и завершение
                                ; работы в случае невыполнения операции
                                ; открыть
        mov     dx,offset msg1  ; DS:DX = адрес сообщения
        mov     cx,msg1_len     ; CX = длина сообщения
        jmp     dump9           ; Начать вывод сообщения

dump4:  mov     fhandle,ax      ; Сохранить дескриптор файла

dump5:                          ; Чтение блока с файловыми данными
        mov     bx,fhandle      ; BX = дескриптор файла
        mov     cx,blksize      ; CX = длина записи
        mov     dx,offset fbuff ; DS:DX = буфер
        mov     ah,3fh          ; Функция 3fh = чтение
        int     21h             ; Переход в MS-DOS

        mov     flen,ax         ; Сохранить общую длину
        cmp     ax,0            ; Конец файла достигнут ?
        jne     dump6           ; Нет, продолжить

        cmp     word ptr fptr,0 ; Это было первое чтение ?
```

Рис. 8.10.

```
        jne     dump8           ; Нет, завершить обычно

                                ; Вывод сообщения и завершение работы
                                ; в случае пустого файла
        mov     dx,offset msg4  ; DS:DX = адрес сообщения
        mov     cx,msg4_len     ; CX = длина сообщения
        mov     dump9           ; Начать вывод сообщения

dump6:                          ; Вывод на экран заголовока
                                ; через каждые 128 байт
        test    fptr,07fh       ; Время выдать заголовок ?
        jnz     dump7           ; Нет, продолжить

                                ; Вывод на экран заголовока
        mov     dx,offset hdg   ; DS:DX = адрес заголовка
        mov     cx,hdg_len      ; CX = длина заголовка
        mov     bx,stdout       ; BX = стандартный вывод
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS

dump7:  call    conv            ; Преобразовать двоичную запись
                                ; в формат ASCII

                                ; Вывод на экран записи
                                ; в формате ASCII
        mov     dx,offset fout  ; DS:DX = адрес вывода
        mov     cx,fout_len     ; CX = длина вывода
        mov     bx,sdout        ; BX = стандартный вывод
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS
        jmp     dump5           ; Переход на чтение следующей записи

dump8:                          ; Закрытие входного файла
        mov     bx,fhandle      ; BX = дескриптор файла
        mov     ah,3eh          ; Функция 3eh = закрыть
        int     21h             ; Переход в MS-DOS

        mov     ax,4c00h        ; Функция 4ch = завершить с
                                ; кодом завершения = 0
        int     21h             ; Переход в MS-DOS

dump9:                          ; Вывод сообщения на устройство
                                ; стандартной ошибки, при этом
                                ; DS:DX =  адрес сообщения
                                ; CX = длина сообщения
        mov     bx,stderr       ; Дескриптор стандартной ошибки
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS

        mov     ax,4c01h        ; Функция 4ch = завершить с
                                ; кодом завершения = 1
        int     21h             ; Переход в MS-DOS

dump    endp

conv    proc    near            ; Преобразование блока данных
                                ; из входного файла
```

Рис. 8.10.

```
        mov     di,offset four  ; Заполнить преобразованную область
        mov     cx,fout_len-2   ; вывода пробелами
        mov     al,blank
        rep stosb

        mov     di,offset fout  ; Преобразовать относительный адрес файла
        mov     ax,fptr         ; в код ASCII для вывода
        call    w2a

        mov     bx,0            ; Начальное значение указателя буфера

conv1:  mov     al,[fbuff+bx]   ; Выбрать байт из буфера
        mov     d,offset foutb  ; Указать область вывода

                                ; Часть файла в формате ASCII ...
                                ; По умолчанию  хранить "."
        mov     byte ptr [di+bx],'.'

        cmp     al,blank        ; Код в диапазоне 20h - 7eh ?
        jb      conv2           ; Перейти, если не алфавитно-цифровой код

        cmp     al,7eh          ; Код в диапазоне 20h - 7eh ?
        ja      conv2           ; Перейти, если не алфавитно-цифровой код


        mov     [di+bx],al      ; Сохранить символ в коде ASCII


conv2:                          ; Часть файла в шестнадцатиричном формате...
        mov     di,offset fouta ; Указать область вывода
        add     di,bx           ; Адрес базы + (относительный.адрес*3)
        add     di,bx
        add     di,bx
        call    b2a             ; Преобразование содержимого байта в
                                ; шестнадцатиричный формат

        inc     bx              ; Перейти к следующей записи
        cmp     bx,flen         ; Входная запись преобразована ?
        jne     conv1           ; Нет, получить другой байт

                                ; Обновление указателя файла
        add     word ptr fptr,blksize

        ret

conv    endp

w2a     proc    near            ; Преобразование слова в шестнадцатиричный
                                ; ASCII-формат. При вызове в регистрах
                                ; AX = значение, DI = адрес строки
                                ; При возврате регистры  AX, DI, CX
                                ; не сохраняются

        push    ax              ; Сохранить копию значения
        mov     al,ah
        call    b2a             ; Преобразовать старший байт

        pop     ax              ; Получить копию назад
```

Рис. 8.10.

```
        call    b2a             ; Преобразовать младший байт
        ret

w2a     endp

b2a     proc    near            ; Преобразование байта в шестнадцатиричный
                                ; ASCII формат. При вызове в регистрах
                                ; AL = двоичное значение,
                                ; DI = адрес строки
                                ; При возврате регистры AX, DI, CX
                                ; не сохраняются

        sub     ah,ah           ; Очистить старший байт
        mov     cl,16
        div     cl              ; Разделить содержимое байта на 16
        call    ascii           ; Частное будет первым
        stosb                   ; символом в коде ASCII
        ret
        mov     al,ah           ;
        call    ascii           ; Остаток будет вторым
        stosb                   ; символом в коде ASCII
        ret

b2a     endp

ascii   proc    near            ; Преобразование значений из AL диапазона
                                ; 0-0fh в шестнадцатиричный ASCII формат
        add     al,'0'          ; Смещение  в диапазоне 0-9
        cmp     al,'9'          ; Текущее значение > 9 ?
        jle     ascii2          ; Нет, перейти
        add     al,'a'-'9'-1    ; Смещение в диапазоне A - F

ascii2: ret                     ; Возвращение в регистре AL
                                ; кода ASCII символа

ascii   endp

argc    proc    near            ; Подсчет аргументов командной строки
                                ; При вызове  ES:BX = командная строка
                                ; При возврате AX = количество аргументов

        push    bx              ; Сохранить исходные значения BX и CX
        push    cx              ; для последующего их использования
        mov     ax,1            ; Установить счетчик >= 1


argc1:  mov     cx,-1           ; Установить флаг = внешний аргумент

argc2:  inc     bx              ; Указать следующий символ
        cmp     byte ptr es:[bx],cr
        je      argc3           ; Завершить, если это код возврата каретки
        cmp     byte ptr es:[bx],blank
        je      argc1           ; Внешний аргумент, если код ASCII пробела
        cmp     byte ptr es:[bx],tab
        je      argc1           ; Внешний аргумент, если код ASCII табуляции

                                ; Обработка кодов отличных от кодов
                                ; пробела и табуляции
```

Рис. 8.10.

```
        jcxz    argc2           ; Перейти, если это внутренний аргумент

        inc     ax              ; Если аргумент не найден, выполнить
                                ; инкремент счетчика
        not     cx              ; Установить флаг = внутренний аргумент
        jmp     argc2           ; и начать обработку следующего символа

argc3:  pop     cx              ; Восстановить исходные значения BX и CX
        pop     bx
        ret                     ; Возвращает AX = число аргументов

argc    endp

argv    proc    near            ; Получение адреса и длины
                                ; аргументов командной строки
                                ; При вызове   EX:BX = командная строка
                                ;              AX    = номер аргумента
                                ; При возврате ES:BX = адрес
                                ;              AX    = длина

        push    cx              ; Сохранить исходные значения CX и DI
        push    di

        or      ax,ax           ; Это аргумент 0 ?
        jz      argv8           ; Да, перейти на получение имени программы

        xor     ah,ah           ; Установить счетчик аргументов

argv1:  mov     cx,-1           ; Установить флаг = внешний аргумент

argv2:  inc     bx              ; Указать следующий символ
        cmp     byte ptr es:[bx],cr
        je      argv7           ; Завершить, если это код возврата каретки
        cmp     byte ptr es:[bx],blank
        je      argv1           ; Внешний аргумент,если это код ASCII пробела
        cmp     byte ptr es:[bx],tab
        je      argv1           ; Внешний аргумент,если это
                                ; код ASCII табуляции

                                ; Обработка кодов отличных от кодов
                                ; пробела и табуляции ...
        jcxz    argv2           ; Перейти, если это внутренний аргумент

        inc     ah              ; Иначе определить счетчик аргументов
        cmp     ah,al           ; Это тот аргумент, который мы ищем ?
        je      argv4           ; Да, начать поиск его длины
        not     cx              ; Нет, установить флаг = внутренний аргумент
        jmp     argv2           ; и начать обработку следующего символа

argv4:                          ; Определение длины найденного
                                ; искомого аргумента ...
        mov     ax,bx           ; Сохранить адрес начала параметра

argv5:  inc     bx              ; Указать следующий символ
        cmp     byte ptr es:[bx],cr
        je      argv6           ; Конец найден,если это код возврата каретки
        cmp     byte ptr es:[bx],blank
```

Рис. 8.10.

```
        je      argv6           ; Конец найден, если это код ASCII пробела
        cmp     byte ptr es:[bx],tab
        jne      argv5          ; Конец найден, если это код ASCII табуляции

argv6:  xchg    bx,ax           ; Установить ES:BX = адрес аргумента
        sub     ax,bx           ; и AX = длина аргумента
        jmp     argvx           ; Вернуться к месту вызова

argv7:  xor     ax,ax           ; Установить AX = 0, аргумент не найден
        jmp     argvx           ; Вернуться к месту вызова

argv8:                          ; Специальная обработка  тех случаев,
                                ; когда число аргументов = 0
        mov     ah,3000h        ; Проверить, это версия DOS 3.0 или старше
        int     21h             ; (Установить AL = 0  в случае DOS 1)
        cmp     al,3
        jb      argv7           ; DOS 1 или 2, вернуть пустой параметр
        mov     es,es:[2ch]     ; Получить из префикса сегмент контекста
        xor     di,di           ; Найти имя программы
        xor     al,al           ; как первый пропуск
        mov     cx,-1           ; после всех переменных контекста ...
        cld
argv9:  rephe   scasb           ; Поиск двух нулей ( SCASW  применяться
        scasb                   ; не может, т.к. возможен нечетный адрес )
        jne     argv9           ; Продолжение цикла, если был одиночный нуль
        add     di,2            ;Пропустить слово счетчика в контексте
        mov     bx,di           ; Сохранить адрес имени программы
        mov     cx,-1           ; Теперь найти ее длину ...
        rephe scasb             ; Поиск другого нулевого байта
        not     cx              ; Преобразовать содержимое CX в длину
        dec     cx
        mov     ax,cx           ; Вернуть длину в AX

argvx:                          ; Общая точка выхода
        pop     di              ; Восстановить исходные значения CX и DI
        pop     cx
        ret                     ; Вернуться к месту вызова

argv    endp

_TEXT   ends

_DATA   segment word public 'DATA'

fname   db      64 dup (0)      ; Буфер спецификации входного файла

fhandle dw      0               ; Дескриптор входного файла из PCDOS

flen    dw      0               ; Общая длина прочитанного

fptr    dw      0               ; Относительный адрес в файле

fbuff   db      blksize dup (?) ; Данные из выходного файла

fout    db      'nnnn'          ; Выходная область преобразованных данных
        db      blank,blank
fouta   db      16 dup ('nn',blank)
        db      blank
```

Рис. 8.10.

```
foutb   db      16 dup (blank),cr,lf
fout_len equ    $-fout

hdg     db      cr,lf           ; Заголовок для каждых 128 байт
        db      7 dup (blank)   ; преобразованных выходных данных
        db      '0 1 2 3 4 5 6 7 '
        db      '8 9 A B C D E F',cr,lf
hdg_len equ     $-hdg
msg1    db      cr,lf
        db      'dump: Файл не найден'
        db      cr,lf
msg1_len equ    $-msg1

msg2    db      cr,lf
        db      'dump: Пропущено имя файла'
        db      cr,lf
msg2_len equ    $-msg2

msg3    db      cr,lf
        db      'dump: Неверная версия MS-DOS '
        db      cr,lf,'$'

msg4    db      cr,lf
        db      'dump: Пустой файл'
        db      cr,lf
msg4_len equ    $-msg4

_DATA   ends

STACK   segment para stack 'STACK'

        db      64 dup (?)

STACK   ends

        end     dump
```

Рис. 8.10.

```
/*
    DUMP.C      Вывод на экран двоичного содержимого файла в шестнадцатерич-
                ном виде и кодах ASCII на устройство стандартного вывода

    Compile:    C>CL DUMP.C

    Usage:      C>DUMP unit:path\filename.ext

    Авторские права (C) 1988 Рей Данкан
*/

#include <stdio.h>
#include <io.h>
#include <fcntl.h>
#define REC_SIZE 16             /* Размер записи входного файла        */

main(int argc, char *argv[])
{
```

Рис. 8.11. Версия программы DUMP.C на языке Си.

```
int fd;                         /* Дескриптор входного файла            */
  int status = 0;               /* Состояние из чтения файла            */
long fileptr = 0L;              /* Относит.адрес текущего файла в байтах*/
char filebuf[REC_SIZE];         /* Данные из файла                      */

if(argc !=2)                    /* Завершить,если пропущено имя файла   */
{   fprintf(stderr,"\ndump: Неверное число параметров\n");
        exit(1);
   }
                                /* Открыть файл в двоичном режиме,
                                   завершить, если операция открыть
                                   выполнена неверно                    */
    if((fd = open(argv[1],O_RDONLY ! O_BINARY) ) == -1
    {   fprintf(stderr, "\ndump: Файл найти не удается %s \n", argv[1]);
        exit(1);
    }

                                /* Чтение и дамп записей до
                                   конца файла                          */
    while((status = read(fd,filebuf,REC_SIZE) ) !=0)
    {   dump_rec(filebuf, fileptr, status);
        fileptr += REC_SIZE;
    }

    close(fd);                  /* Закрыть входной файл                 */
    exit(0);                    /* Возврат кода успешного завершения    */
}

/*
    Вывести на стандартный вывод запись (16 байт) в шестнадцатеричном
    представлении  и кодах ASCII
*/

dump_rec(char *filebuf, long fileptr, int length)
{
    int i;                      /* Индекс текущей записи                */

    if(fileptr % 128 == 0)      /*Если нужно,вывести на экран заголовок*/
        printf("\n\n          0 1 2 3 4 5 6 7 8 9 A B C D E F");

    printf("\n%041X ",fileptr); /* Вывести на экран относит.адрес файла*/
                                /* Вывести на экран шестнадцатеричный
                                   эквивалент каждого байта файла       */
    for(i = 0; i < length; i++)
        printf(" %02X",(unsigned char) filebuf[i]);

    if(length !=16)             /* Пробелы, если неполная запись        */
        for (i=); i<(16-length);i++) printf("  ")

                                /* Вывести на экран код ASCII содержимого
                                   байта файла                          */
    printf("  ");
    for(i = 0; i < legth: i++)
    {   if(filebuf[i] < 32 !! filebuf[i] > 126) putchar('.');
        else putchar(filebuf[i]);
    }
 }
```

Рис. 8.11.

- Программа производит выделение из хвоста команды в префиксе сегмента программы его составляющих: дисковода, пути и имени файла.
- Ради скорости программа использует буферизованный ввод-вывод.
- Программа пересылает сообщение об ошибках на устройство стандартной ошибки.
- Программа пересылает обычный программный вывод на устройство стандартного вывода, так что выходной дамп появится по умолчанию на системной консоли, но может перенаправляться на другие символьно-ориентированные устройства (такие, как устройство построчной печати) или в файл.

Эти же особенности включены и в вариант программы на языке Си, но, кроме того, здесь есть особенности использования во время работы программы библиотеки программ на языке Си.

Вариант программы DUMP на языке ассемблера содержит несколько подпрограмм, которые могут оказаться полезными для вас при создании других программ. К ним относятся:

Подпрограмма	**Описание**
argc	Возвращает количество аргументов командной строки
argv	Возвращает адрес и длину конкретного аргумента командной строки
w2a	Преобразует двоичное представление слова (16 бит) в шестнадцатиричное ASCII-представление, требуемое для операции вывода
b2a	Преобразует двоичное представление байта (8 бит) в шестнадцатиричное ASCII-представление, требуемое для операции вывода
ascii	Преобразует двоичное представление 4 бит в один символ в коде шестнадцатиричного ASCII-представления

Интересно сравнить эти эквивалентные программы. Программа на языке Си содержит только 77 строк, в то время как программа на ассемблере – 436 строк. Очевидно, что исходная программа на языке Си менее сложна и ее проще модифицировать. С другой стороны, если необходимо заботиться о размерах и быстродействии, то файл DUMP.EXE, сгенерированный компилятором Си, имеет длину 8563 байт, в то время как файл DUMP.EXE, полученный из программы на языке ассемблера,– только 1294 байт и выполняется в 2 раза быстрее, чем программа на языке Си.

ГЛАВА 9. ТОМА И КАТАЛОГИ

Каждый файл в системе MS-DOS однозначно идентифицируется своим именем и своим местоположением. Местоположение в свою очередь определяется двумя компонентами: логическим дисководом, на котором располагается файл, и каталогом на этом дисководе, в котором можно найти имя файла.

Логические дисководы обозначаются буквой с двоеточием за ней (например, A:). Число логических дисководов в системе не обязательно совпадает с числом физических дисководов: дисководы с большими жесткими дисками разделяют на два или более логических дисковода. Ключевой характеристикой логического дисковода является наличие на нем самостоятельной системы файлов; другими словами, дисковод содержит один или более каталогов, ноль или более целых файлов, а также всю информацию, необходимую для определения местонахождения файлов и каталогов и того, какие области диска свободны, а какие уже заняты.

Каталоги – это просто списки. Каждая запись в каталоге включает имя, размер, начальный адрес, атрибуты, дату и время последней модификации файла или другого

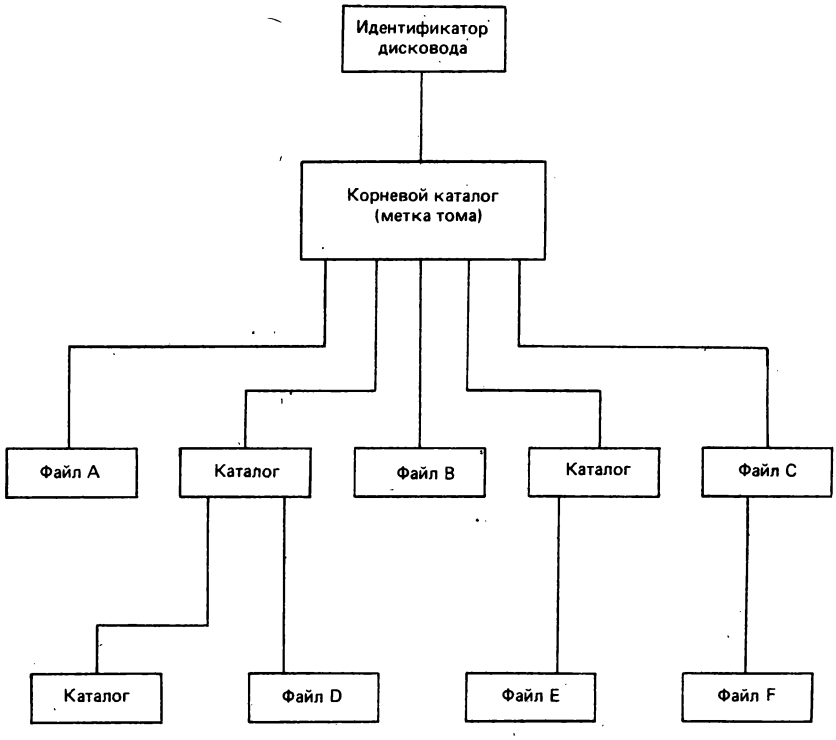


Рис. 9.1. Структура системы файлов MS-DOS.

каталога, находящихся на диске. Детальная информация о расположении каждого блока данных, назначенного файлу или каталогу, хранится в специальной управляющей области диска, называемой «таблица размещения файлов» (file allocation table, FAT). (Детальное описание внутреннего формата каталогов и FAT приведено в гл. 10.)

Каждый диск в принципе может содержать каталоги двух типов: корневой каталог и все остальные каталоги. Корневой каталог всегда присутствует на диске, причем максимально допустимое число записей в нем определяется в процессе форматирования диска и не может быть изменено. Подкаталоги корневого каталога, наличие которых на диске не обязательно, могут иметь любой уровень вложенности и любой размер (рис. 9.1). Это и есть иерархическая, или древовидная, структура, которая упоминалась в предыдущих главах. У каждого каталога имеется имя, кроме корневого каталога, который обозначается символом «\».

MS-DOS сама определяет, какой дисковод является «текущим» для системы, и использует этот дисковод, если в спецификации файла отсутствует обозначение дисковода в явной форме. Точно так же MS-DOS хранит имя «текущего каталога» для каждого логического дисковода. Вы можете выбрать любой конкретный каталог на дисководе, перечислив последовательно – либо от корневого каталога, либо относительно текущего каталога – все каталоги, ведущие к его местоположению на древовидной структуре. Этот список имен каталогов, разделенных символами «\», носит название «путь». Строка, состоящая из обозначений логического дисковода, полного пути (от корневого каталога) и имени файла с расширением, перечисленных в указанном порядке, образует полностью квалифицированное имя файла, однозначно описывающее файл.

УПРАВЛЕНИЕ ДИСКОВОДАМИ И КАТАЛОГАМИ

С помощью команд DIR, CHDIR (CD), MKDIR (MD) и RMDIR (RD) в интерактивном режиме можно просматривать, выбирать, создавать и удалять каталоги диска. Для выбора нового текущего дисковода следует ввести букву, обозначающую требуемый дисковод, и вслед за ней двоеточие. Для того чтобы и прикладные программы могли управлять дисководами и каталогами, в MS-DOS предусмотрены следующие функции, реализуемые с помощью прерывания Int 21H:

Функция	Действие
0EH	Выбрать текущий дисковод
19H	Получить текущий дисковод
39H	Создать каталог
3AH	Удалить каталог
3BH	Выбрать текущий каталог
47H	Получить текущий каталог

Обе функции управления дисководами принимают или возвращают двоичный код дисковода: 0 – представляет дисковод A, 1 – дисковод B и т. д. Это не совпадает с обозначениями, используемыми большинством других функций MS-DOS, когда 0 указывает на текущий дисковод, 1 – на дисковод A и т. д.

Три первые функции управления каталогами из приведенного списка требуют строку кодов ASCIIZ, описывающую путь к указанному каталогу. Так же как и в случае функций открытия и закрытия файла, использующих дескрипторы, адрес строки ASCIIZ передается через регистры DS:DX. Выход осуществляется со сброшенным флагом переноса, если функция выполнена успешно, и с установленным флагом в противном случае; в регистре AX возвращается код ошибки. Функции управления каталогами могут не выполниться по многим причинам, из которых самая распространенная – указание в пути несуществующего элемента.

Последняя функция приведенного списка, Int 21H с функцией 47H, позволяет получить в виде строки ASCIIZ путь к текущему каталогу на указанном или подразумеваемом дисководе. В строке, формируемой MS-DOS, отсутствует обозначение дисковода вместе с лидирующим символом «\». Прерывание Int 21H с функциями 47H и 19H обычно используется для построения полностью квалифицированного имени файла. Удобство такого имени заключается в том, что оно остается правильным, даже если пользователь изменяет текущий диск или каталог.

В разд. 2 этой книги «Справочник по функциям MS-DOS» дано подробное описание функций управления дисководами и каталогами.

Поиск в каталогах

Когда вы запрашиваете операцию открытия файла, вы в неявной форме выполняете поиск в каталоге. MS-DOS изучает каждую запись в каталоге, чтобы найти ту, которая содержит данное вами в качестве аргумента имя файла; если файл найден, MS-DOS копирует определенную информацию из каталога в структуру данных, которая затем используется для управления операциями чтения или записи файла. Поэтому если вам требуется проверить, существует ли конкретный файл, вы можете выполнить операцию открытия и посмотреть, успешно ли она выполнилась. (Если успешно, то вам, конечно, следует выполнить и операцию закрытия, чтобы не расходовать понапрасну дескрипторы.)

Иногда вам может понадобиться усложнить операцию поиска в каталоге диска.

Например, вы хотите найти все файлы с определенным расширением, файл с конкретным атрибутом или имена подкаталогов данного каталога. Хотя местоположение каталогов диска и особенности находящихся в них записей являются, естественно, аппаратно-зависимыми (например, значение поля, описывающего начальный адрес файла, определяется физическим форматом диска), MS-DOS предоставляет функции, позволяющие исследовать каталог диска аппаратно-независимым образом.

Чтобы успешно осуществлять поиск в каталоге диска, следует разобраться в двух типах операций поиска, выполняемых средствами MS-DOS. К первому типу относится функция «поиск первого». Эта функция, получив спецификацию файла,– возможно, с символами шаблона групповой операции – отыскивает в интересующем нас каталоге первый файл, удовлетворяющий условиям заданной спецификации. Если такой файл найден, функция заполняет буфер запрашивающей программы информацией о файле; если файл не найден, функция возвращает флаг ошибки.

Программа может вызвать функцию поиска второго типа, называемую «поиск следующего», только после успешного «поиска первого». Если спецификация файла, переданная функции «поиск первого», содержала символы шаблона групповой операции и по крайней мере один файл, удовлетворяющий условиям шаблона, существует, программа может вызвать «поиск следующего» столько раз, сколько требуется для обнаружения всех оставшихся файлов, описываемых использованным шаблоном. Как и «поиск первого», «поиск следующего» возвращает информацию о найденных файлах в буфер, выделенный запрашивающей программой. Если MS-DOS более не может найти файлов, удовлетворяющих шаблону, «поиск следующего» возвращает флаг ошибки.

Как и почти во всех остальных операциях, MS-DOS предоставляет два параллельных набора функций поиска в каталоге:

Действие	Функция типа FCB	Функция типа дескриптора
Поиск первого	11H	4EH
Поиск следующего	12H	4FH

Функции типа FCB осуществляют поиск по имени файла и расширению (оба элемента могут содержать шаблоны групповых операций) в текущем каталоге текущего или указанного дисковода. Функции типа дескриптора, с другой стороны, позволяют программе выполнять поиск в любом каталоге на любом дисководе независимо от того, какой каталог является текущим.

Операции поиска, использующие обычные FCB, находят только обычные файлы. Операции, использующие расширенные FCB, или функции типа дескриптора, могут учитывать атрибуты файла. К операциям поиска имеют отношение следующие биты атрибутов:

Бит	Значение
0	файл «только для чтения»
1	скрытый файл
2	системный файл
3	метка тома
4	каталог
5	требуется архивация (устанавливается после модификации файла)

Остальные биты параметра атрибутов функции поиска должны быть равны 0. Если какие-либо из перечисленных выше битов атрибутов установлены, функция поиска возвращает все обычные файлы плюс все файлы с указанными атрибутами. Если,

однако, установлен бит атрибута метки тома, функция поиска выполняет специальные действия, описанные ниже в этой главе. Заметьте, что, установив бит 4, вы можете включить в операцию поиска каталоги, как если бы они представляли собой обычные файлы.

Функции поиска в каталогах как типа FCB, так и типа дескриптора требуют, чтобы адрес дисковой области передачи (disk transfer area, DTA) был установлен (с помощью прерывания Int 21H с функцией 1AH) перед вызовом операции «поиск первого» и указывал на рабочий буфер для использования MS-DOS. Адрес DTA нельзя изменять между вызовами «поиска первого» и «поиска следующего». MS-DOS, обнаружив удовлетворяющий условиям поиска файл, заносит в буфер информацию о файле, а затем при выполнении очередной операции «поиска следующего» исследует содержимое буфера, чтобы определить, где продолжить поиск. Форматы данных, возвращаемых в буфер функциями типа FCB и типа дескриптора, различаются, поэтому перед попыткой интерпретировать содержимое буфера ознакомьтесь с его подробным описанием, приведенным в разд. 2 этой книги «Справочник по функциям MS-DOS».

На рис. 9.2 и 9.3 приведены эквивалентные программные фрагменты, реализующие поиск в данном каталоге всех файлов с расширением .ASM. В первом примере использованы функции поиска типа FCB (Int 21H с функциями 11H и 12H), во втором – функции типа дескриптора (Int 21H с функциями 4EH и 4FH). (Для того чтобы в приведенных листингах не было различий, не связанных с сутью дела, обе программы для вывода информации о найденных файлах используют функции записи типа дескриптора с дескриптором стандартного вывода.)

```
start:                          ; Установить адрес DTA -
                                ; буфер для функций поиска
        mov     dx,seg buff     ; DS:DX = адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        mov     ah,1ah          ; Функция 1ah = установка DTA
        int     21h             ; Переход в MS-DOS
                                ; Поиск первого соответствия...
        mov     dx,offset fcb   ; DS:DX = адрес FCB
        mov     ah,11h          ; Функция 11h = поиск первого
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Есть хоть одно соответствие?
        jnz     exit            ; Нет, на выход

disp:                           ; На новую строку...
        mov     dx,offset crlf  ; DS:DX = строка CR-LF
        mov     cx,2            ; CX = длина строки
        mov     bx,1            ; BX = стандартный указатель вывода
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS

                                ; Вывод найденного файла
        mov     dx,offset buff+1; DS:DX = имя файла
        mov     cx,11           ; CX = длина
        mov     bx,1            ; BX = стандартный указатель вывода
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS
```

Рис. 9.2. Пример операции типа FCB поиска в каталоге с использованием прерывания Int 21H с функциями 11H и 12H.

Программа выводит имена всех файлов с расширением ASM, входящих в текущий каталог.

```
                                ; Поиск следующего соответствия
        mov     dx,offset fcb   ; DS:DX = адрес FCB
        mov     ah,12h          ; Функция 12h = поиск следующего
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Есть еще соответствия?
        jz      disp            ; Да, на вывод имени файла

exit:                           ; Точка окончательного выхода
        mov     ax,4c00h        ; Фукция 4ch = завершение
                                ; Код возврата = 0
        int     21h             ; Переход в MS-DOS
        .
        .
        .
crlf    db      0dh,0ah         ; Строка ASCII возврат каретки -
                                ; перевод строки (CR-LF)

fcb     db      0               ; Дисковод = текущий
        db      8 dup ('?')     ; Имя файла = шаблон
        db      'ASM'           ; Расширение = ASM
        db      25 dup (0)      ; Остаток FCB = 0

buff    db      64 dup (0)      ; Для результатов поиска
```

Рис. 9.2.

```
start:                          ; Установить адрес DTA -
                                ; буфер для функций поиска
        mov     dx,seg buff     ; DS:DX = адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        mov     ah,1ah          ; Функция 1ah = установка DTA
        int     21h             ; Переход в MS-DOS

                                ; Поиск первого соответствия...
        mov dx,offset fname     ; DS:DX = групповое имя файла
        mov     cx,0            ; CX = атрибут обычного файла
        mov     ah,4eh          ; Функция 4eh = поиск первого
        int     21h             ; Переход в MS-DOS
        jc      exit            ; Выход,если соответствий нет

disp:                           ; На следующую строку...
        mov     dx,offset crlf  ; DS:DX = строка CR-LF
        mov     cx,2            ; CX = длина строки
        mov     bx,1            ; BX = стандартный дескриптор вывода
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS
                                ; Найдем длину файла...
        mov     cx,0            ; CX будет счетчиком символов
                                ; DS:SI = начало имени
        mov     si,offset buff+30

disp1:  lodsb                   ; Получить следующий символ
        or      al,al           ; Это нулевой символ?
```

Рис. 9.3. Пример операции типа дескриптора поиска в каталоге с использованием прерывания Int 21H с функциями 4EH и 4FH.

Программа также выводит имена всех файлов с расширением ASM, входящих в текущий каталог.

```
        jz      disp2           ; Да, нашли конец строки
        inc     cx              ; Нет, посчитаем символы
        jmp     disp1           ; и получим следующий

disp2:                          ; Вывод найденного файла...
                                ; CX уже содержит длину
                                ; DS:DX = имя файла
        mov     dx,offset buff+30
        mov     bx,1            ; BX = стандартный указатель вывода
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS
                                ; Поиск следующего файла...
        mov     ah,4fh          ; Функция 4fh = поиск следующего
        int     21h             ; Переход в MS-DOS
        jnc     disp            ; Переход, если найден еще файл

exit:                           ; Точка окончательного выхода
        mov     ax,4c00h        ; Фукция 4ch = завершение
                                ; Код возврата = 0
        int     21h             ; Переход в MS-DOS
        .
        .
crlf    db      0dh,0ah         ; Строка ASCII возврат каретки -
                                ; перевод строки (CR-LF)

fname   db      '*.ASM',0       ; Имя файла для поиска
                                ; соответствия - строка ASCIIZ

buff    db      64 dup (0)      ; Для результатов поиска
```

Рис. 9.3.

Перемещение файлов

Функция переименования файлов (Int 21H с функцией 56H), добавленная в версии 2.0 MS-DOS, обладает малоизвестной возможностью перемещения файла из одного каталога в другой. Функция имеет два параметра в кодах ASCIIZ: «старое» и «новое» имена файла. Если старый и новый пути различаются, MS-DOS перемещает файл; если различаются имена или расширения файлов, MS-DOS переименовывает файл. MS-DOS может выполнить оба этих действия в одном и том же функциональном вызове.

Естественно, что и старый, и новый каталог должны быть на одном дисководе, потому что фактически данные, образующие файл, совсем не перемещаются; лишь информация, описывающая файл, изымается из одного каталога и помещается в другой. Функция 56H не выполняется, если две строки кодов ASCIIZ включают отличающиеся имена логических дисководов, если файл имеет атрибут «только для чтения» или если уже существует файл с тем же именем и местоположением, как и «новый» файл.

Функция переименования файлов типа FCB (Int 21H с функцией 17H) работает только в текущем каталоге и не может быть использована для перемещения файлов.

МЕТКИ ТОМА

Поддержка меток тома была впервые добавлена в MS-DOS версии 2.0. Метка тома представляет собой необязательное имя длиной от 1 до 11 символов, которое пользователь назначает диску в процессе операции FORMAT. Вывести значение метки тома на экран можно с помощью команд DIR, TREE, CHKDSK и VOL. Начиная с MS-DOS версии 3.0, с помощью команды LABEL можно добавить, вывести на экран

или изменить метку после операции форматирования. В MS-DOS версии 4.0 программа FORMAT, кроме того, назначает каждому форматируемому диску псевдослучайный 32-разрядный двоичный идентификационный код; этот код можно вывести на экран, но нельзя изменить.

Важно понимать разницу между томами и дисководами. Метка тома связана с конкретной запоминающей средой. Идентификатор дисковода (например, A) связан с физическим устройством, на котором может быть смонтирована запоминающая среда. В случае несъемных жестких дисков среду, связанную с идентификатором дисковода, нельзя сменить (откуда и название). В случае гибких дисков или другой сменной среды диск, для обращения к которому используется данный идентификатор дисковода, может иметь любую метку тома или не иметь никакой.

В результате метка тома не может заменить идентификатор логического дисковода и ее нельзя использовать в качестве части обозначения пути для идентификации файла. В действительности в системе MS-DOS версии 2.0 метка тома во внутренних операциях вообще не используется. В MS-DOS версии 3.0 и выше драйвер диска может использовать метку тома для определения того, не сменил ли пользователь диск при открытом файле; эта проверка, однако, не является обязательной и выполняется не во всех системах.

Метки тома MS-DOS реализованы в виде записей специального вида в корневом каталоге диска. Запись содержит информацию о дате и времени и характеризуется значением байта атрибута, равным 8 (т.е. установленным битом 3). За исключением

```
buff    db      64 dup (?)      ; Получает результаты поиска

xfcb    db      0ffh            ; Флаг расширенного FCB
        db      5 dup (0)       ; Зарезервировано
        db      8               ; Байт атрибута метки тома
        db      0               ; Код дисковода (0 = текущий)
        db      11 dup ('?')    ; Групповое имя и расширение
        db      25 dup (0)      ; Остаток FCB (не используется)
        .
        .
        .
                                ; Установить адрес DTA -
                                ; буфер для функций поиска
        mov     dx,seg buff     ; DS:DX = адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        mov     ah,1ah          ; Функция 1ah = установка DTA
        int     21h             ; Переход в MS-DOS

                                ; Теперь ищем метку...
                                ; DS:DX = расширенный FCB
        mov     dx,offset xfcb
        mov     ah,11h          ; Функция 11h = поиск первого
        int     21h             ; Переход в MS-DOS
        cmp     al,0ffh         ; Поиск успешен?
        je      no_label        ; Переход,если нет метки тома
        .
        .
        .
```

Рис. 9.4. Поиск метки тома в MS-DOS версии 2 с использованием расширенного блока управления файлом.

В случае успешного поиска метка тома возвращается в buff, форматированная в полях имени файла и расширения имени файла расширенного FCB.

```
buff    db      64 dup (?)      ; Получает результаты поиска
wildcd  db      '*.*',0         ; Групповое имя файла (строка ASCIIZ)
        .
        .
        .
                                ; Установить адрес DTA -
                                ; буфер для функций поиска
        mov     dx,seg buff     ; DS:DX = адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        mov     ah,1ah          ; Функция 1ah = установка DTA
        int     21h             ; Переход в MS-DOS

                                ; Теперь ищем метку ...
                                ; DS:DX = строка ASCIIZ
        mov     dx,offset wildcd
        mov     cx,8            ; CX = атрибут метки тома
        mov     ah,4eh          ; Функция 4eh = поиск первого
        int     21h             ; Переход в MS-DOS
        jc      no_label        ; Переход,если нет метки тома
        .
```

Рис. 9.5. Поиск метки тома в MS-DOS версии 3 с использованием функций поиска типа дескриптора.

В случае успешного поиска (флаг переноса возвращается сброшенным), имя тома размещается по адресу buff + 1EH в форме строки ASCIIZ.

атрибута, запись с меткой тома идентична записи каталога с информацией о файле, который был создан, но ни разу не заполнялся никакими данными, и вы можете работать с метками тома с помощью функций Int 21H точно так же, как и с файлами. Однако для метки тома характерны некоторые особенности, проявляющиеся на разных уровнях:

- Если вы создаете метку тома на отформатированном ранее диске, MS-DOS всегда помещает ее в корневой каталог независимо от того, какой каталог является текущим.
- Диск может содержать только одну метку тома; попытки создать дополнительные метки (даже с другими именами) не увенчаются успехом.
- MS-DOS всегда выполняет поиск меток тома в корневом каталоге независимо от того, какой каталог является текущим, и не возвращает при этом обычные файлы.

В MS-DOS версии 2.0 поддержка меток тома не включена в полной мере в функции типа дескриптора, и для работы с метками тома следует использовать расширенные FCB. На рис. 9.4 приведен пример программы, осуществляющей поиск метки тома в корневом каталоге текущего дисковода. С помощью расширенных FCB и функции переименования файла (Int 21H с функцией 17H) можно изменить метку тома, однако в MS-DOS версии 2.0 нельзя пытаться удалить существующую метку тома, используя Int 21H с функцией 13H, потому что эта операция может повредить FAT диска непредсказуемым образом.

В MS-DOS версий 3.0 и выше, воспользовавшись прерыванием Int 21H с функцией 3CH и значением атрибута, равным 8, можно создать метку тома, а функция «поиска первого» типа дескриптора (4EH) позволяет получить существующую метку тома на логическом дисководе (рис. 9.5). Тем не менее для изменения метки тома по-прежнему надо использовать расширенные FCB.

ГЛАВА 10. ВНУТРЕННЯЯ СТРУКТУРА ДИСКА

Диски MS-DOS организованы по довольно жесткой схеме, которую нетрудно понять и, следовательно, использовать при работе с ними. Хотя вам, возможно, никогда не понадобится непосредственно обращаться к специальным управляющим областям диска, понимание их внутренней структуры приведет к лучшему пониманию поведения и функционирования MS-DOS в целом.

С точки зрения прикладного программиста, MS-DOS представляет дисковые устройства как логические тома, обозначаемые кодом дисковода (A, B, C и т. д.) и содержащие необязательное имя тома, корневой каталог и от нуля до любого числа дополнительных каталогов и файлов. MS-DOS защищает программиста от физических характеристик среды, предоставляя целый набор средств обслуживания диска с помощью прерывания Int 21H. Используя эти средства, программист может создавать, открывать, читать, записывать, закрывать и удалять файлы единообразными действиями безотносительно к размеру и скорости диска, числу головок чтения-записи, количеству дорожек и т. д.

Фактически запросы прикладной программы на операции с файлами проходят два уровня преобразования перед тем, как воплотиться в физическую передачу данных между дисковым устройством и оперативной памятью.

1. Внутри себя MS-DOS рассматривает каждый логический том, будь это целое физическое устройство, например гибкий диск, или только часть жесткого диска, как непрерывную последовательность логических секторов, начинающуюся с сектора 0. (Том логического диска может быть реализован и на других типах запоминающих

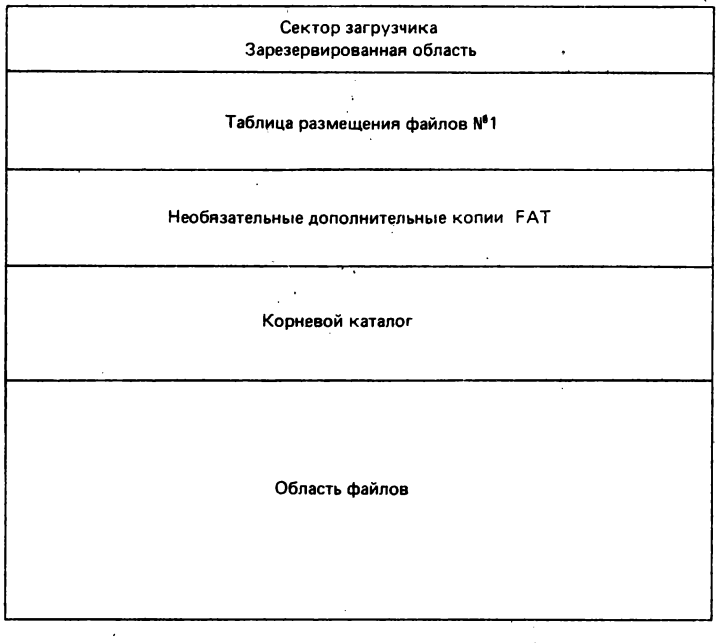


Рис. 10.1. Карта типичного логического тома MS-DOS.

Сектор загрузчика (логический сектор 0) содержит идентификатор изготовителя, блок параметров BIOS (BPB) и начальный загрузчик диска. Оставшиеся секторы содержат необязательную зарезервированную область, одну или более копий таблицы размещения файлов, корневой каталог и область файлов.

устройств. Например, электронные диски отображают дисковую структуру на область запоминающего устройства с произвольной выборкой.) MS-DOS преобразует запросы прикладной программы на управление файлами Int 21H в запросы на пересылку логических секторов, используя при этом информацию, полученную из каталогов тома и таблиц размещения. (В тех редких случаях, когда это необходимо, программы могут обращаться к логическим секторам непосредственно с помощью Int 25H и Int 26H.)

2. После этого MS-DOS передает запросы на логические секторы драйверу диска, который отображает их на действительные физические адреса (головка, дорожка и сектор). Драйверы диска являются в высшей степени аппаратно-зависимыми и ради максимального повышения быстродействия всегда пишутся на языке ассемблера. В большинстве версий MS-DOS драйвер IBM-совместимых гибких и жестких дисков встроен в модуль BIOS (IO.SYS) и всегда загружается в процессе инициализации системы; включив в файл CONFIG.SYS соответствующую директиву DEVICE, можно установить дополнительные драйверы для IBM-несовместимых дисков.

Каждый логический том MS-DOS включает помимо области файлов еще несколько управляющих областей фиксированного размера (рис. 10.1). Размер каждой управляющей области зависит от ряда факторов, например размера тома и версии программы FORMAT, с помощью которой был инициализирован том, но вся информация, необходимая для интерпретации структуры конкретного логического тома, находится на самом томе в секторе загрузчика.

СЕКТОР ЗАГРУЗЧИКА

Логический сектор 0, известный как *сектор загрузчика*, содержит всю важную информацию о характеристиках диска (рис. 10.2). Первый байт сектора всегда содержит команду безусловного перехода микропроцессоров 80 × 86 – либо команду нормального внутрисегментного перехода JMP (код операции 0E9H) с 16-разрядным смещением, либо команду «короткого» перехода JMP (код операции 0EBH) с 8-разрядным смещением, за которой стоит команда NOP (код операции 90H). Если в первом байте ни то ни другое, значит, диск либо не форматирован вообще, либо не форматирован для работы с MS-DOS. (Конечно, присутствие кода операции JMP еще не гарантирует, что диск форматирован для MS-DOS.)

Вслед за начальной командой JMP расположено 8-байт поле, зарезервированное фирмой Microsoft для идентификации фирмы-изготовителя оборудования. Программа форматирования диска, специализированная под конкретные модификации компьютера, контроллера диска или магнитного носителя, заполняет это поле именем изготовителя компьютера и фирменным номером версии MS-DOS.

В байтах с 0BH до 17H размещается третий важный компонент сектора загрузчика – блок параметров BIOS (BIOS parameter block, BPB). (В версиях MS-DOS 3.0 и выше в сектор загрузчика входят дополнительные поля.) Эта структура данных описывает физические характеристики диска и позволяет драйверу устройства вычислить правильный физичекий адрес на диске по данному номеру логического сектора; в этой структуре также содержится информация, используемая MS-DOS и разнообразными системными утилитами для вычисления адреса и размера каждой управляющей области диска (таблиц размещения файлов и корневого каталога).

Последний элемент сектора загрузчика – это программа дискового начального загрузчика. Дисковый начальный загрузчик обычно считывается в память с помощью программы, хранящейся в ПЗУ начальной загрузки и запускаемой автоматически при включении компьютера. Программе в ПЗУ едва хватает ума для сброса головок дисковода (перевода их на дорожку 0), чтения первого физического сектора в ОЗУ, начиная с установленного заранее адреса и перехода на считанную программу.

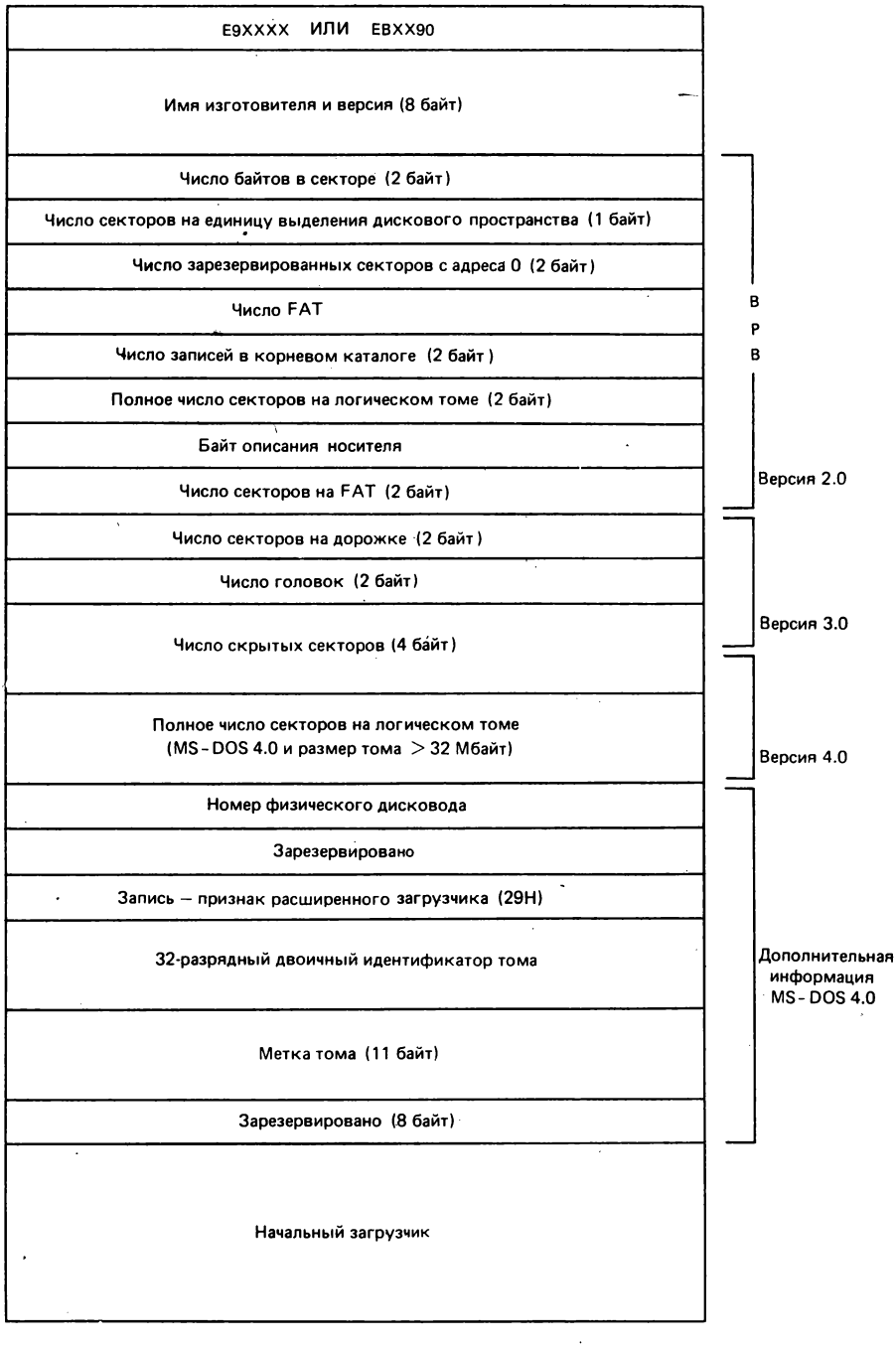


Рис. 10.2. Карта сектора загрузчика диска MS-DOS.

Обратите внимание на команду JMP по адресу 0, поле идентификации изготовителя, блок параметров BIOS (совместимый с MS-DOS версии 2.0) в байтах 0BH–17H, три дополнительных словных поля для MS-DOS версии 3, поле длиной в 2 слова для числа секторов, 32-разрядный двоичный идентификатор тома 1D для MS-DOS версии 4.0, а также программу начальной загрузки.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   .
0000  EB 34 90 49 42 4D 20 20 33 2E 33 00 02 02 01 00  .4.IBM  3.3.....
0010  02 70 00 D0 02 FD 02 00 09 00 02 00 00 00 00 00  .p..............
0020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 12  ................
0030  00 00 00 00 01 00 FA 33 C0 8E D0 BC 00 7C 16 07  .......3.....|..
      .
      .
      .
01C0  0D 0A 44 69 73 6B 20 42 6F 6F 74 20 66 61 69 6C  ..Disk Boot fail
01D0  75 72 65 0D 0A 00 49 42 4D 42 49 4F 20 20 43 4F  ure...IBMBIO  CO
01E0  4D 49 42 49 44 4F 53 20 20 43 4F 4D 00 00 00 00  MIBMDOS  COM....
01F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA  ..............U.
```

Рис. 10.3. Шестнадцатеричный дамп участков сектора загрузчика (дорожка 0, головка 0, сектор 1) гибкого диска PC-DOS версии 3.3.

Этот сектор содержит идентификацию изготовителя, копию блока параметров BIOS, описывающего носитель, и программу начальной загрузки, которая считывает BIOS в память и передет ей управление. (См. также рис. 10.2 и 10.4.)

```
jmp     $+54            ; Переход на начальный загрузчик
nop

db      'IBM  3.3'      ; Идентификация изготовителя

                        ; Блок параметров BIOS
dw      512             ; Байт на сектор
db      2               ; Секторов на кластер
dw      1               ; Зарезервировано секторов
db      2               ; Число FAT
dw      112             ; Записи корневого каталога
dw      720             ; Всего секторов
db      0fdh            ; Байт описания носителя
dw      2               ; Секторов на FAT

dw      9               ; Секторов на дорожку
dw      2               ; Число головок
dd      0               ; Скрытых секторов
```

Рис. 10.4. Результат дизассемблирования участка сектора загрузчика, показанного на рис. 10.3.

Дисковый загрузчик посложнее. Он вычисляет физический адрес на диске, где начинается область файлов, считывает в память файлы, содержащие операционную систему, и передает управление модулю BIOS по адресу 0070:0000H (см. гл. 2).

На рис. 10.3 и 10.4 показаны шестнадцатеричный дамп и результат дизассемблирования участков сектора загрузчика гибкого диска PC-DOS 3.3.

ЗАРЕЗЕРВИРОВАННАЯ ОБЛАСТЬ

Сектор загрузчика является частью зарезервированной области, занимающей один или несколько секторов. Размер этой области указан в слове для зарезервированных секторов в BPB по адресу 0EH в секторе загрузчика. Учтите, что число в поле BPB включает и сам сектор загрузчика, так что, если это число равно 1 (как это имеет место

на дискетах IBM PC), длина зарезервированной области фактически составляет 0 секторов.

ТАБЛИЦА РАЗМЕЩЕНИЯ ФАЙЛОВ

При создании или расширении файла MS-DOS назначает ему группы секторов диска из области файлов, причем число секторов в группе составляет степень числа 2. Эти группы служат единицами выделения дискового пространства и называются кластерами. Число секторов в каждом кластере для данного носителя записано в BPB по адресу 0DH в секторе загрузчика. Ниже приведены примеры размеров кластеров.

Тип диска	Степень 2	Число секторов в кластере
Дискета 5,25″ 180 Кбайт	0	1
Дискета 5,25″ 360 Кбайт	1	2
Жесткий диск PC/AT	2	4
Жесткий диск PC/XT	3	8

Таблица размещения файлов (FAT) разделена на поля, однозначно соответствующие кластерам диска, предназначенным для размещения файлов. Эти поля в MS-DOS версий 1 и 2 имеют размер 12 бит, а в версии 3 и выше могут составлять 12 или 16 бит в зависимости от размера носителя (12 бит, если диск содержит менее 4087 кластеров, и 16 бит в противном случае).

Первые два поля в FAT всегда зарезервированы. На IBM-совместимых носителях первые 8 бит первой зарезервированной записи FAT содержат копию байта описания носителя из BPB в секторе загрузчика. Второй, третий и четвертый (при его наличии) байты, составляющие остаток двух первых зарезервированных полей FAT, всегда содержат 0FFH. Байты описания носителя формата IBM в настоящее время имеют следующие значения:

Байт описания	Носитель	Версия MS-DOS, в которой введена поддержка
0F0H	Дискета 3,5", 2 стороны, 18 секторов	3.3
0F8H	Жесткий диск	2.0
0F9H	Дискета 5,25", 2 стороны, 15 секторов	3.0
	Дискета 3,5", 2 стороны, 9 секторов	3.2
0FCH	Дискета 5,25", 1 сторона, 9 секторов	2.0
0FDH	Дискета 5,25", 2 стороны, 9 секторов	2.0
	Дискета 8", 1 сторона, одинарная плотность	
0FEH	Дискета 5,25", 1 сторона, 8 секторов	1.0
	Дискета 8", 1 сторона, одинарная плотность	
	Дискета 8", 2 стороны, двойная плотность	
0FFH	Дискета 5,25", 2 стороны, 8 секторов	1.1

Остальные записи FAT описывают использование соответствующих им кластеров диска. Содержимое полей FAT интерпретируется следующим образом:

Значение	Интерпретация
(0) 000H	Кластер свободен
(F) FF0–(F) FF6H	Зарезервированный кластер
(F) FF7H	Дефектный кластер, если не часть цепочки
(F) FF8–(F) FFFH	Последний кластер файла
(X) XXX	Следующий кластер файла

Каждая запись в каталоге, соответствующая тому или иному файлу, включает номер первого кластера, назначенного этому файлу, который используется как точка входа в FAT. Начиная с этой точки, каждая запись FAT содержит номер следующего кластера в файле, вплоть до метки последнего кластера.

По выбору изготовителя компьютера MS-DOS поддерживает одну или две идентичные копии FAT на каждом томе. Каждый раз при расширении файла или модификации каталога MS-DOS изменяет одновременно обе копии. Если при обращении к сектору в FAT фиксируется ошибка чтения, MS-DOS пытается читать другие копии до успешного выполнения операции чтения или исчерпания всех копий. Таким образом, если одна из копий FAT становится нечитаемой из-за износа носителя или программного сбоя, наличие других копий позволяет все же спасти содержимое диска. Программа CHKDSK в процессе выполнения процедуры проверки целостности диска сравнивает все (обычно две) копии FAT, чтобы убедиться в их читаемости и согласованности.

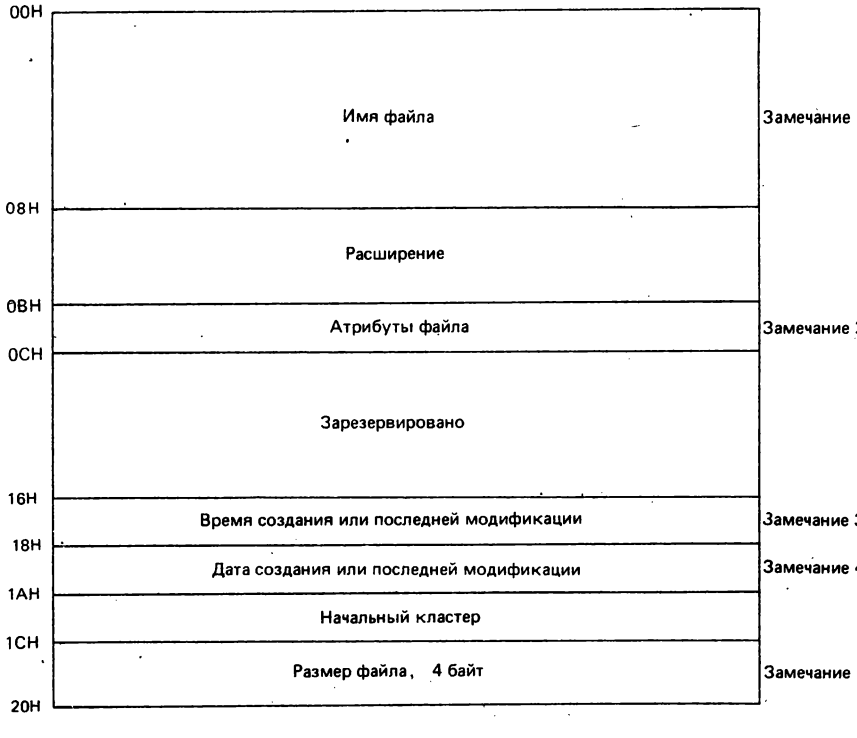


Рис. 10.5. Формат записи в каталоге диска. Полная длина 32 байт (20H байт).

КОРНЕВОЙ КАТАЛОГ

Вслед за таблицами размещения файлов расположена область, известная в MS-DOS версий 2.0 и выше как корневой каталог. (В MS-DOS версии 1 корневой каталог был единственным каталогом диска.) Корневой каталог содержит 32-байт записи, которые описывают файлы, другие каталоги и необязательную метку тома (рис. 10.5). Запись, начинающаяся с байта со значением E5H, свободна для повторного использования; она представляет удаленный файл или каталог. Запись, начинающаяся с нулевого байта, характеризует логический конец каталога; и эта запись, и все последующие никогда не использовались.

Замечания к рис. 10.5

1. Первый байт поля записи каталога, отведенного под имя файла, может содержать следующую специальную информацию:

Код	Значение
00H	Запись каталога ни разу не использовалась; конец занятой части каталога
05H	Первый символ имени файла фактически E5H
2EH	Запись для корневого или родительского каталога. Если следующий байт также равен 2EH, поле кластера содержит номер кластера родительского каталога (и 0, если родительским является корневой каталог)
E5H	Файл был удален

2. Байт атрибутов записи каталога содержит следующие атрибуты файла:

Бит	Значение
0	Файл «только для чтения»; попытка открыть файл для записи или удаления не удастся
1	Скрытый файл; исключается из нормальной процедуры поиска
2	Системный файл; исключается из нормальной процедуры поиска
3	Метка тома; может существовать только в корневом каталоге
4	Каталог; исключается из нормальной процедуры поиска
5	Бит архива; устанавливается после модификации файла
6	Зарезервирован
7	Зарезервирован

3. Поле времени кодируется следующим образом:

Биты	Содержимое
00H–04H	Двоичное количество 2-секундных единиц времени (0–29, что соответствует 0–58 с)
05H–0AH	Двоичное число минут (0–59)
0BH–0FH	Двоичное число часов (0–23)

4. Поле даты кодируется следующим образом:

Биты	Содержимое
00H–04H	День месяца (1–31)
05H–08H	Месяц (1–12)
09H–0FH	Год (относительно 1980)

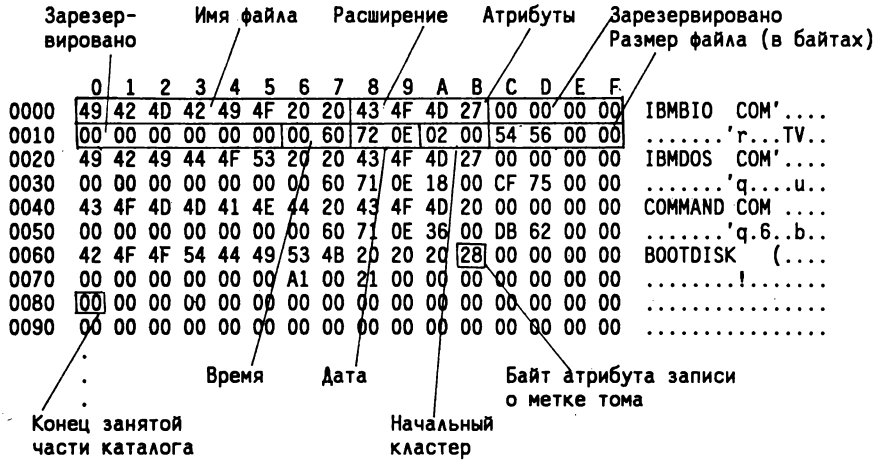


Рис. 10.6. Шестнадцатеричный дамп первого сектора корневого каталога на диске PC-DOS версии 3.3, содержащего три системных файла и метку тома.

5. Поле размера файла рассматривается как 4-байт целое; в память записываются сначала два младших байта, затем два старших.

Корневой каталог имеет ряд отличий. Его размер и положение фиксированы; их определяет программа FORMAT в процессе инициализации диска. Указанные характеристики корневого каталога можно найти в BPB сектора загрузчика. Если диск является загружаемым, две первые записи корневого каталога всегда описывают файлы, содержащие BIOS и ядро MS-DOS. Программа начальной загрузки использует эти записи для загрузки операционной системы в память и ее запуска.

На рис. 10.6 показан участок шестнадцатеричного дампа первого сектора корневого каталога загружаемой дискеты PC-DOS 3.3.

Область файлов

Все остальное пространство тома после корневого каталога известно как область файлов. MS-DOS рассматривает эту область как совокупность кластеров, каждый из которых в зависимости от формата диска содержит один или более логических секторов. Каждому кластеру соответствует запись в ТРФ, описывающая его текущее состояние: свободен, зарезервирован, назначен файлу или не подлежит использованию (в результате повреждения магнитного носителя). Из-за того что первые два поля FAT зарезервированы, первому кластеру в области файлов присвоен номер 2. При расширении файла MS-DOS версии 1 или 2 просматривает FAT от начала, пока не встретит свободный кластер (характеризуемый нулевым значением поля FAT); в это поле записывается метка последнего кластера, а поле предыдущего кластера цепочки изменяется так, чтобы оно указывало на новый последний кластер. MS-DOS версий 3.0 и выше начинает просмотр FAT от кластера, который был назначен последним; это уменьшает фрагментацию и общее время доступа.

Все каталоги, кроме корневого, представляют собой просто файлы специального типа. Пространство для них выделяется из области файлов, а их содержимое составляют 32-байт записи, совпадающие по формату с записями корневого каталога и описывающие файлы и другие каталоги. Записи каталога, описывающие другие каталоги, характеризуются байтом атрибутов с установленным битом 4, нулем в поле

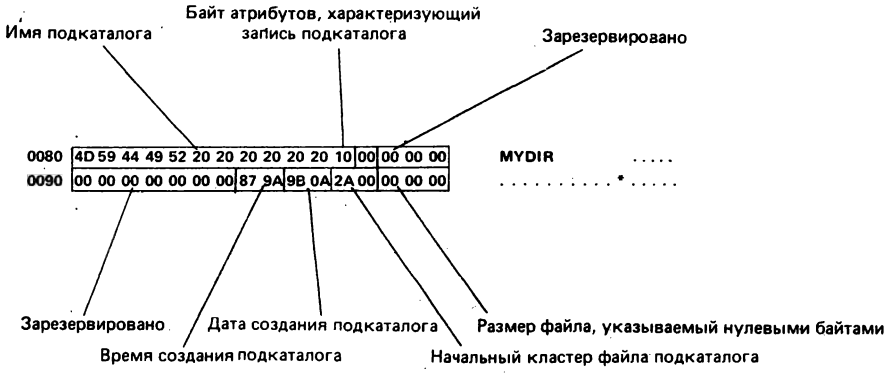


Рис. 10.7. Часть корневого каталога диска MS-DOS с записью для подкаталога, названного MYDIR.

Бит 4-й байта атрибутов установлен, поле кластера указывает на первый кластер файла подкаталога, поля даты и времени заполнены, но длина файла равна 0.

длины файла, а также датой и временем создания каталога (рис. 10.7). Поле первого кластера указывает, естественно, на первый кластер в области файлов, принадлежащий этому каталогу. (Остальные кластеры каталога можно найти только путем прослеживания их цепочки в FAT.)

Все каталоги, кроме корневого, содержат две специальные записи каталога с именами «.» и «..». MS-DOS размещает эти записи при создании каталога, и их нельзя удалить. Запись с точкой является синонимом текущего каталога; поле кластера этой записи указывает на кластер, в котором находится каталог. Запись с двумя точками является синонимом родительского каталога (каталога, находящегося на дереве каталогов непосредственно над текущим); ее поле кластера указывает на первый кластер родительского каталога. Если родительским является корневой каталог, поле кластера записи с двумя точками содержит 0 (рис. 10.8).

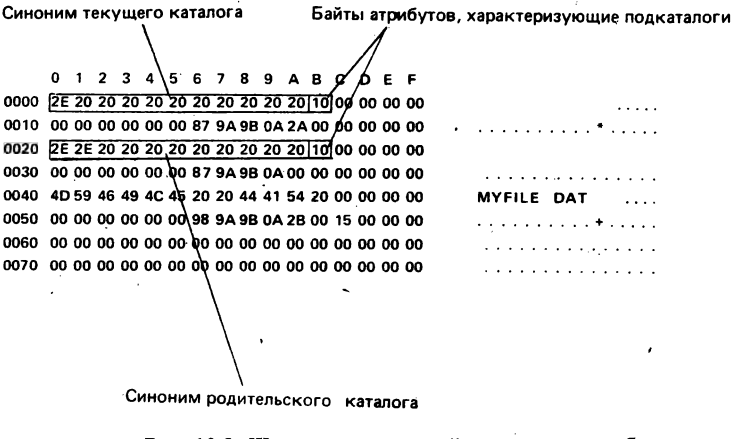


Рис. 10.8. Шестнадцатеричный дамп первого блока каталога MYDIR.

Обратите внимание на записи с одной и двумя точками. Этот каталог содержит точно один файл MYFILE.DAT.

ИССЛЕДОВАНИЕ ТАБЛИЦЫ РАЗМЕЩЕНИЯ ФАЙЛОВ

Теперь, когда мы рассмотрели структуру диска, можно использовать эти знания для нахождения позиции FAT по номеру кластера.

Если FAT содержит 12-бит записи, используйте следующую процедуру:

1. С помощью записи каталога найдите начальный кластер рассматриваемого файла.
2. Умножьте номер кластера на 1,5.
3. Используйте целую часть произведения в качестве относительного адреса в FAT и перешлите слово, расположенное по этому адресу, в регистр. Не забудьте, что позиция FAT может пересекать физическую границу сектора диска.
4. Если произведение представляет собой целое число, наложите на содержимое регистра маску 0FFFH (операция логического И).
5. В противном случае сдвиньте содержимое регистра вправо на 4 бит командой логического сдвига.
6. Если в результате образуется число от 0FF8H до 0FFFH, файл больше не содержит кластеров. В противном случае результат представляет собой номер следующего кластера в файле.

На дисках объемом не менее 4087 кластеров и форматированных в MS-DOS версий 3.0 и выше записи FAT содержат 16 бит и выделение из таблицы номера кластера заметно упрощается.

1. С помощью записи каталога найдите начальный кластер рассматриваемого файла.
2. Умножьте номер кластера на 2.
3. Используйте произведение в качестве относительного адреса в FAT и перешлите слово, расположенное по этому адресу, в регистр.
4. Если в результате образуется число от 0FF8H до 0FFFH, файл больше не содержит кластеров. В противном случае результат представляет собой номер следующего кластера в файле.

Для преобразования номера кластера в номер логического сектора вычтите 2, умножьте результат на число секторов в кластере и прибавьте номер логического сектора, с которого начинается область данных (этот номер можно определить с помощью информации в BPB).

В качестве примера определим местоположение на диске файла IBMBIO.COM, характеристики которого содержатся в первой записи на рис. 10.6. Прежде всего нам нужна некоторая информация из BPB, входящего в сектор загрузчика диска (см. рис. 10.3 и 10.4). Из BPB можно узнать, что диск имеет следующие характеристики:

512 байт на сектор
2 сектора на кластер
2 сектора в FAT
2 FAT
112 записей в корневом каталоге

С помощью информации, содержащейся в BPB, можно вычислить номера логических секторов, с которых начинаются управляющие области диска и область файлов, записав полученные данные в таблицу:

Область	**Длина в секторах**	**Номера секторов**
Сектор загрузчика	1	00H
2FAT * 2 сектора в FAT	4	01H–04H

Область	Длина в секторах	Номера секторов
112 записей каталога * (32 байт/запись)/(512 байт/сектор)	7	05H–0BH
Полное число секторов, занятых начальным загрузчиком, обеими FAT и корневым каталогом	12	

Таким образом, номер первого сектора области данных составляет 12 (0CH).

Слово с относительным адресом 01AH в записи каталога для файла IBMBIO.COM дает нам номер начального кластера этого файла: кластер 2. Для определения номера логического сектора первого блока этого файла следует воспользоваться описанной выше процедурой.

1. Номер кластера – 2 = 2 – 2 = 0.
2. Умножаем на число секторов в кластере = 0 × 2 = 0.
3. Прибавляем номер логического сектора начала области файлов = 0 + 0CH = = 0CH.

Таким образом, вычисленный номер сектора, с которого начинается файл IBMBIO.COM, равен 0CH, что в точности совпадает с ожидаемым, учитывая, что программа FORMAT всегда помещает системные файлы в смежные секторы в самом начале области данных.

Попробуем теперь проследить цепочку IBMBIO.COM по таблице размещения файлов (рис. 10.9 и 10.10). Это будет утомительное занятие, но детальное понимание процесса необходимо. В реальной программе мы сначала прочитаем сектор загрузчика с помощью вызова Int 25H, затем по содержимому BPB вычислим адрес FAT и, наконец, прочитаем FAT в память, снова использовав вызов Int 25H.

Из записи каталога для IBMBIO.COM мы уже знаем, что первым кластером в файле является кластер 2. Для исследования записи в FAT, соответствующей этому кластеру, умножаем номер кластера на 1,5, что дает 0003H – относительный адрес FAT, и извлекаем слово по этому адресу (получая 4003H). Поскольку при умножении номера кластера на 1,5 получается целое число, накладываем не содержимое слова, извлеченного из FAT, маску 0FFFH, что дает число 3 – номер второго кластера, назначенного файлу.

Для исследования записи в FAT, соответствующей третьему кластеру, умножаем 3 на 1,5 получаем 4,5 и извлекаем слово по относительному адресу 0004H (получая 0040H). Поскольку при умножении 3 на 1,5 получается нецелое число, сдвигаем слово

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0000  FD FF FF 03 40 00 05 60 00 07 80 00 09 A0 00 0B  ....@..'........
0010  C0 00 0D E0 00 0F 00 01 11 20 01 13 40 01 15 60  ......... ..@..'
0020  01 17 F0 FF 19 A0 01 1B C0 01 1D E0 01 1F 00 02  ................
0030  21 20 02 23 40 02 25 60 02 27 80 02 29 A0 02 2B  ! .#@.%'.'..)..+
      .
      .
      .
```

Рис. 10.9. Шестнадцатеричный дамп первого блока таблицы размещения файлов (дорожка 0, головка 0, сектор 2) для диска PC-DOS версии 3.3, корневой каталог которого показан на рис. 10.6.

Обратите внимание на то, что первый байт FAT является байтом описания носителя: 5,25 дюйма, 2 стороны, 9-секторная дискета.

```
getfat  proc    near            ; Извлекает поле FAT
                                ; для данного кластера
                                ; Вызов AX = номер кластера
                                ;       DS:BX = адрес FAT
                                ; Возврат AX = поле FAT
                                ; Остальные регистры без изменения

        push    bx              ; Сохранить используемые регистры
        push    cx
        mov     cx,ax
        shl     ax,1            ; Кластер * 2
        add     ax,cx           ; Кластер * 3
        test    ax,1
        pushf                   ; Сохранить остаток в флаге Z
        shr     ax,1            ; Кластер * 1,5
        add     bx,ax
        mov     ax,[bx]
        popf                    ; Кластер * 1,5 целое число?
        jnz     getfat1         ; Нет, переход
        and     ax,0fffh        ; Да, изолировать 12 младших бит
        jmp     getfat2
getfat1:mov     cx,4            ; Сдвинуть слово вправо на 4 бит
        shr     ax,cx
getfat2:pop     cx              ; Восстановить регистры и выйти
        pop     bx
        ret
getfat  endp
```

Рис. 10.10. Процедура на языке ассемблера, реализующая доступ к таблице размещения файлов (предполагаются 12-бит поля FAT).

Получив номер кластера, процедура возвращает содержимое записи FAT для этого кластера в регистре AX. В этом простом примере игнорируется то обстоятельство, что записи FAT могут пересекать границы секторов.

вправо на 4 бит и получаем число 4, которое представляет номер третьего кластера, назначенного файлу IBMBIO.COM.

Продолжая описанные действия, прослеживаем цепочку номеров кластеров в FAT, пока не дойдем до кластера (в данном случае с номером 23), у которого запись в FAT содержит число 0FFFH, представляющее собой метку конца файла в FAT с 12-бит записями.

Мы выяснили, что файл IBMBIO.COM содержит кластеры с номерами от 2 до 23 (02H–17H), откуда можно вычислить, что файлу назначены логические секторы от 0CH до 38H. Конечно, последний кластер может быть заполнен данными только частично; чтобы найти, какая доля последнего кластера заполнена данными, надо остаток файла (длина файла содержится в записи каталога) разделить на число байтов, составляющих кластер.

РАЗДЕЛЫ ЖЕСТКОГО ДИСКА

Помимо уже обсуждавшейся структуры логических томов жесткий диск имеет еще один уровень организации – разделы. Утилита FDISK делит жесткий диск на один или больше разделов, состоящих из целого числа цилиндров. Каждый раздел может содержать независимую файловую систему и, более того, собственную копию операционной системы.

Первый физический сектор на жестком диске (дорожка 0, головка 0, сектор 1) содержит главную загрузочную запись, состоящую из следующих элементов:

Байты	Содержимое
000–1BDH	Зарезервированы
1BE–1CDH	Дескриптор раздела № 1
1CE–1DDH	Дескриптор раздела № 2
1DE–1EDH	Дескриптор раздела № 3
1EE–1FDH	Дескриптор раздела № 4
1FE–1FFH	Слово признаков (AA55H)

Дескрипторы разделов в главной загрузочной записи определяют размер, расположение и тип каждого раздела следующим образом:

Байты	Содержимое
00H	Флаг активности (0 – не загружаемый, 80H – загружаемый)
01H	Начальная головка
02H–03H	Начальные цилиндр/сектор
04H	Тип раздела 00H не используется 01H файловая система с FAT, 12-бит записи FAT 04H файловая система с FAT, 16-бит записи FAT 05H расширенный раздел 06H «гигантский раздел» (MS-DOS версий 4.0 и выше)
05H	Конечная головка
06H–07H	Конечные цилиндр/сектор
08H–0BH	Начальный сектор раздела относительно начала диска
0CH–0FH	Длина раздела в секторах

Флаг активности, указывающий на то, что раздел является загружаемым, может быть установлен в каждый момент времени только в одном разделе.

Рис. 10.11. Шестнадцатеричный дамп участка главного блока жесткого диска, форматированного в PC-DOS версии 3.3.

Этот диск содержит два раздела. Первый раздел имеет 16-бит FAT и отмечен как «активный», чтобы показать, что раздел содержит загружаемую копию PC-DOS. Второй раздел является «расширенным». Третий и четвертый разделы в этом примере не используются.

MS-DOS рассматривает разделы типов 1, 4 и 6 как обычные логические тома и в процессе начальной загрузки системы назначает им их собственные идентификаторы дисководов. Раздел типа 5 может включать много логических томов и имеет специальную расширенную загрузочную запись, описывающую каждый том. Утилита FORMAT инициализирует разделы жесткого диска MS-DOS, создавая внутри раздела файловую систему (запись загрузчика, таблицу размещения файлов, корневой каталог и область файлов) и размещая (по требованию) в файловой системе загружаемую копию операционной системы.

На рис. 10.11 показан шестнадцатеричный дамп участка главного блока жесткого диска, форматированного в MS-DOS версии 3.3. Дамп иллюстрирует дескрипторы разделов для обычного раздела в 16-бит FAT и для расширенного раздела.

ГЛАВА 11. УПРАВЛЕНИЕ ПАМЯТЬЮ

Современные версии MS-DOS обслуживают до 1 Мбайт непрерывного пространства ОЗУ. На IBM PC и совместимых с ними компьютерах память, занимаемая MS-DOS и другими программами, начинается с адреса 0000H и может достигать адреса 09FFFFH; эту область памяти размером 640 кбайт иногда называют обычной памятью (conventional memory). Память за указанным пределом адресов зарезервирована для аппаратных драйверов, хранящихся в ПЗУ, буферов регенерации видеотерминалов и других подобных применений. Компьютеры, не относящиеся к числу IBM-совместимых, могут использовать другое распределение памяти.

Пространство основной памяти, контролируемое MS-DOS, делится на два основных участка:

- Область операционной системы
- Область транзитных программ[1] (транзитная область)

Область операционной системы начинается с адреса 0000H, другими словами, она занимает самую начальную часть памяти. Эта область содержит таблицу векторов прерываний, собственно операционную систему вместе с ее таблицами и буферами, любые дополнительные устанавливаемые драйверы, описанные в файле CONFIG.SYS, а также резидентную часть интерпретатора команд COMMAND.COM. Объем памяти, занимаемый областью операционной системы, зависит от версии MS-DOS, числа дисковых буферов, размеров установленных драйверов устройств и т. д.

Транзитная область, иногда называемая ареной памяти, включает всю память, оставшуюся за областью операционной системы. Арена памяти распределяется динамическими блоками, носящими название элементов арены. В каждый элемент арены входит специальная управляющая структура, называемая заголовком арены, причем все заголовки арены связаны цепочкой. В составе MS-DOS имеются три функции Int 21H, позволяющие программам распределять, изменять размер и освобождать блоки памяти из транзитной области:

Функция	Действие
48H	Выделить блок памяти
49H	Освободить блок памяти
4AH	Изменить размер блока памяти

Система MS-DOS сама использует эти функции при загрузке программ с диска по

[1] Транзитные (т. е. нерезидентные в памяти) программы называют также внешними. – *Прим. перев.*

запросу COMMAND.COM или другой программы. Функция EXEC, представляющая собой загрузчик программ MS-DOS, вызывает Int 21H с функцией 48H для выделения блока памяти под окружение загружаемой программы и другого блока под саму программу и префикс программного сегмента. Затем программа считывается с диска в выделенную область памяти. Когда программа завершается, MS-DOS вызывает Int 21H с функцией 49H для освобождения всей памяти, занимаемой программой.

Транзитные программы также могут использовать функции управления памятью MS-DOS для динамического распределения памяти, имеющейся в транзитной области. Правильное использование этих функций является одним из важнейших критериев корректного поведения программы в рамках MS-DOS. Корректные программы (well-behaved programs) с максимальной вероятностью окажутся переносимыми в будущие версии операционной системы и с минимальной вероятностью приведут к конфликтам с другими процессами при использовании многозадачных интерфейсов с пользователем, таких, как Microsoft Windows.

ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ РАСПРЕДЕЛЕНИЯ ПАМЯТИ

Функции распределения памяти используются в двух основных случаях.

- Для сокращения пространства памяти, выделенного первоначально программе, чтобы освободить место для загрузки и выполнения другой программы, выполняемой под управлением первой.
- Для динамического распределения дополнительной памяти, требуемой программой, и освобождения этой памяти, когда надобность в ней отпадет.

СОКРАЩЕНИЕ НАЧАЛЬНОГО РАСПРЕДЕЛЕНИЯ ПАМЯТИ

Хотя многие прикладные программы MS-DOS попросту предполагают, что вся память находится в их распоряжении, такой подход является пережитком, оставшимся от MS-DOS версии 1 (и от системы CP/M), которые поддерживали в каждый момент времени только один активный процесс. Корректные программы MS-DOS никогда не модифицируют память, которой они фактически не владеют, и обязательно освобождают память, которая им не нужна.

К сожалению, работая под управлением современных версий MS-DOS, трудно предугадать заранее, сколько памяти потребуется программе. Оказывается, объем памяти, выделяемый программе при первой загрузке в память, зависит от двух факторов:

- Типа файла, из которого загружается программа.
- Количества свободной памяти, имеющейся в транзитной области.

MS-DOS всегда выделяет программам, загружаемым из файлов .COM (образа памяти), весь свободный блок памяти максимального размера, имеющийся в транзитной области. Поскольку программы .COM не имеют заголовка файла, с помощью которого MS-DOS передается информация о сегментах и использовании памяти, MS-DOS предполагает наихудший случай и дает программе все, что может. MS-DOS загружает программу, если в памяти имеется свободный блок, достаточно большой, чтобы разместить программу плюс 256 байт для префикса программного сегмента плюс 2 байт для стека. Программа .COM, получив управление, должна определить, хватит ли ей памяти для выполнения своих функций.

Для загрузки программ из файлов .EXE MS-DOS придерживается более сложных правил выделения памяти. Во-первых, естественно, в транзитной области должен быть достаточно большой блок, чтобы разместить объявленные сегменты программы, данных и стека. Кроме этого компоновщик устанавливает в заголовке файла .EXE два

```
        .
        .
        org     100h

main    proc    near            ; Точка входа из MS-DOS
                                ; DS, ES = адрес префикса
                                ; программного сегмента

        mov     sp,offset stk   ; Программа COM должна переслать
                                ; стек в область сохранения

                                ; Освободить лишнюю память...
        mov     ah,4ah          ; Функция 4Ah =
                                ; Изменить размер блока памяти
                                ; BX = сколько параграфов сохранить
        mov     bx,(offset stk - offset main + 10FH) / 16
        int     21h             ; Переход в MS-DOS
        jc      error           ; Если ошибка изменения размера,
                                ; переход


        .
main    endp

        .
        .
        .

        dw      64 dup (?)      ; Новая область стека
stk     equ     $               ; Новая база стека

        end     main            ; Определяет точку входа
```

Рис. 11.1. Пример программы .COM, освобождающий избыток памяти после получения управления от MS-DOS.

Вызывается Int 21H с функцией 4AH, причем ES указывает на префикс программного сегмента, а в BX содержится число параграфов, требуемое для выполнения программы. В этом случае новый размер блока памяти, занимаемого программой, вычисляется как сумма размера образа программы и размера префикса (256 байт), округленная до следующего параграфа. Программы .EXE используют аналогичные строки.

поля, информирующие MS-DOS о требованиях памяти со стороны программы. Первое поле, MIN_ALLOC, определяет минимальное число параграфов, которое требуется программе в дополнение к параграфам для сегментов программы, данных и стека. Второе поле, MAX_ALLOC, определяет максимальное число параграфов дополнительной памяти, которые программа могла бы использовать при их наличии.

Загружая файл .EXE, MS-DOS сначала пытается выделить число параграфов, указанное в MAX_ALLOC, плюс число параграфов, требуемое для самой программы. Если такой объем свободной памяти отсутствует, MS-DOS выделяет программе самый большой свободный блок при условии, что выделенный объем не меньше указанного в MIN_ALLOC, плюс размер образа программы. Если это условие не соблюдается, программа выполняться не может.

Программы .COM и .EXE после загрузки могут использовать прерывание Int 21H с функцией 4AH (изменить размер блока памяти) для освобождения памяти, не требуемой в настоящий момент. Обычно это делается сразу после передачи управления от MS-DOS программе путем вызова функции изменения размера блока, причем адрес префикса программного сегмента передается через регистр ES, а число параграфов, необходимое для выполнения программы,– через регистр BX (рис. 11.1).

ДИНАМИЧЕСКОЕ РАСПРЕДЕЛЕНИЕ ДОПОЛНИТЕЛЬНОЙ ПАМЯТИ

Если корректная программа нуждается в дополнительном пространстве памяти – например, для буферов ввода-вывода или массива промежуточных результатов, – она может вызвать прерывание Int 21H с функцией 48H (выделить блок памяти), указав требуемое число параграфов. При наличии достаточно большого блока нераспределенной памяти MS-DOS возвращает адрес сегмента базы назначенной области и сбрасывает флаг переноса (0), что указывает на успешное выполнение функции.

```
            .
            .
            .
            mov     ah,48h          ; 48h = выделить блок памяти
            mov     bx,0800h        ; 800h параграфов = 32K
            int     21h             ; Переход в MS-DOS
            jc      error           ; Переход, если не распределено
            mov     buff_seg,ax     ; Сохранить сегмент распре-
                                    ; деленного блока
            .
            .
            .
            mov     es,buff_seg     ; ES:DI = адрес блока
            xor     di,di
            mov     cx,08000h       ; Записать 32768 байт
            mov     al,0ffh         ; Заполнить буфер единицами
            cld
            rep     stosb           ; Теперь быстро заполним
            .
            .
            .
            mov     cx,08000h       ; Длина записываемого, байт
            mov     bx,handle       ; Дескриптор открытого ранее файла
            push    ds              ; Сохранить наш сегмент данных
            mov     ds,buff_seg     ; Пусть DS:DX = адрес буфера
            mov     dx,0
            mov     ah,40h          ; Функция 40h = запись
            int     21h             ; Переход в MS-DOS
            pop     ds              ; Восстановить наш сегмент данных
            jc      error           ; Переход, если ошибка записи
            .
            .
            .
            mov     es,buff_seg     ; ES = сегмент выделенного ранее блока
            mov     ah,49h          ; Освободить блок памяти
            int     21h             ; Переход в MS-DOS
            jc      error           ; Переход, если не освобохдено
            .
error:      .
            .
handle      dw      0               ; Указатель файла
buff_seg    dw      0               ; Сегмент распределенного блока
            .
            .
            .
```

Рис. 11.2. Пример динамического распределения памяти.

Программа требует от MS-DOS блок памяти размером 32 Кбайт, заполняет его единицами, записывает на диск, а затем освобождает.

Если нераспределенных блоков достаточного размера не имеется, MS-DOS устанавливает флаг переноса (1), возвращает в регистр AX код ошибки, а в регистр BX – размер (в параграфах) максимального доступного блока (рис. 11.2). В этом случае распределения памяти не происходит. Программа может воспользоваться значением, возвращенным в регистр BX, чтобы решить, возможно ли ее дальнейшее выполнение в таком «вырожденном» варианте, с меньшей памятью.

Диспетчер памяти MS-DOS, просматривая цепочку заголовков арены с целью удовлетворения запроса на выделение памяти, может воспользоваться одним из следующих алгоритмов.

Поиск от начала: используется элемент арены с наименьшим адресом, достаточный по величине для удовлетворения запроса.

Оптимальный поиск: используется наименьший по величине элемент арены, удовлетворяющий запросу, независимо от его местоположения.

Поиск от конца: используется элемент арены с максимальным адресом, достаточный по величине для удовлетворения запроса.

Если выбранный элемент арены превышает затребованный размер, MS-DOS делит его на две части: блок затребованного размера, который выделяется программе, выдавшей запрос Int 21H с функцией 48H, и нераспределенный блок, содержащий остаток памяти.

MS-DOS по умолчанию использует алгоритм поиска от начала. Однако в MS-DOS версий 3.0 и выше прикладная программа может изменить алгоритм с помощью прерывания Int 21H с функцией 58H.

После того как программа закончила работу с выделенным блоком, она должна вызвать прерывание Int 21H с функцией 49H для освобождения блока. Если программа этого не делает, MS-DOS при завершении программы автоматически освободит всю выделенную ей память.

ЗАГОЛОВКИ АРЕНЫ

К настоящему времени фирма Microsoft не объявила официально внутреннюю структуру заголовков арены. Возможно, это сделано для того, чтобы программисты не пытались выполнять распределение памяти непосредственно, а только с помощью специально предназначенных для этого функций MS-DOS.

Структура заголовков арены в версиях MS-DOS 2 и 3 идентична. Они занимают 16 байт (один параграф) и расположены непосредственно перед той областью памяти, которую они контролируют (рис. 11.3). Заголовок арены содержит следующую информацию:

- Байт, указывающий, является ли заголовок промежуточным или последним элементом во всей цепочке заголовков.
- Слово, указывающее, является ли контролируемая заголовком область памяти свободной или она уже принадлежит программе (последнем случае слово указывает на префикс программного сегмента).
- Слово, указывающее размер (в параграфах) управляемой области памяти (элемента арены).

Каждый раз, когда программа выдает запрос на выделение, модификацию или освобождение блока памяти, а также при завершении обычной или дочерней программы, MS-DOS исследует цепочку заголовков арены. При обнаружении дефектных блоков или разрыва цепочки MS-DOS выводит аварийное сообщение

Memory allocation error (Ошибка распределения памяти)

и останавливает систему.

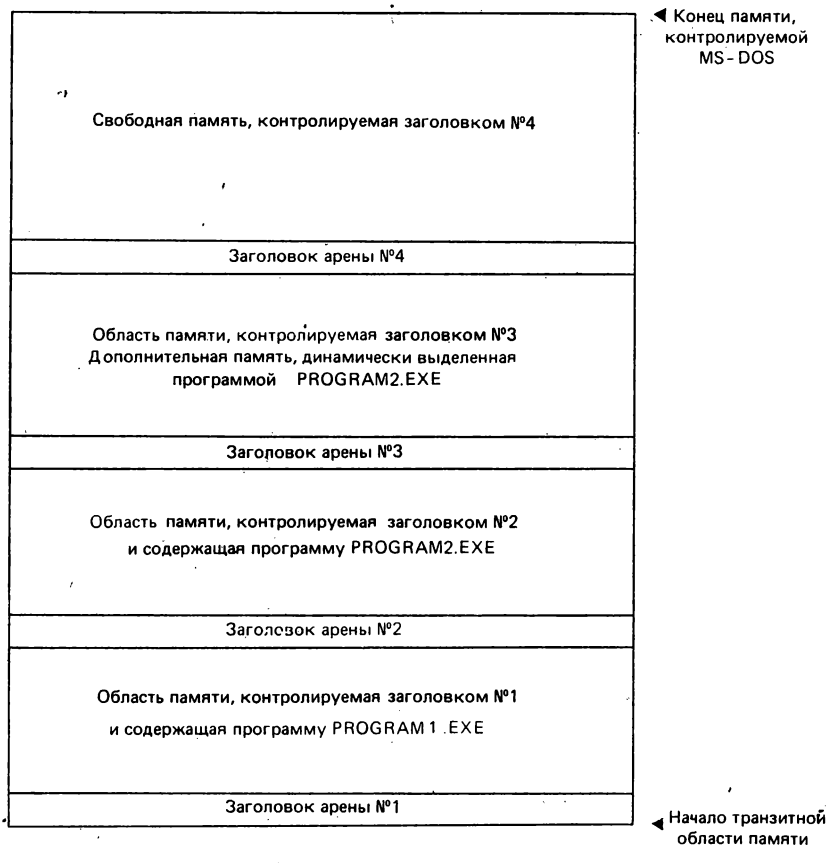


Рис. 11.3. Примерная схема заголовков арены MS-DOS и транзитной области памяти.
Для повышения наглядности схемы блоки окружения и связанные с ними заголовки на рисунке опущены.

В примере, приведенном на рис. 11.3, интерпретатор COMMAND.COM предварительно загрузил в транзитную область программу PROGRAM1.COM и, поскольку она загружена из файла .COM, выделил ей все пространство в транзитной области, контролируемое заголовком арены № 1. Затем программа PROGRAM1.COM воспользовалась прерыванием Int 21H с функцией 4AH (изменить размер блока памяти) для сокращения выделенной ей памяти до величины, действительно требуемой для выполнения программы, после чего с помощью функции EXEC (Int 21H с функцией 4BH) загрузила и выполнила дочернюю программу PROGRAM2.EXE. Функция EXEC получила достаточное пространство памяти, контролируемое заголовком арены № 2, и загрузила в него программу PROGRAM2.EXE. Этой программе в свою очередь потребовалась дополнительная память для записи некоторых промежуточных результатов, поэтому она вызвала прерывание Int 21H с функцией 48H (выделить блок памяти), чтобы получить область памяти, контролируемую заголовком арены № 3. Самый старший заголовок арены (№ 4) контролирует всю оставшуюся транзитную область, не распределенную никакой программе.

ДОПОЛНИТЕЛЬНАЯ ПАМЯТЬ LOTUS/INTEL/MICROSOFT

Когда на рынке появились первые персональные компьютеры фирмы IBM с системой MS-DOS, предел 640 Кбайт, установленный IBM для пространства ОЗУ, непосредственно контролируемой MS-DOS, представлялся невероятно огромным. Но по мере того, как увеличивались возможности и размер MS-DOS, а популярные прикладные программы наращивали свою мощность, эти 640 Кбайт стали казаться тесноватыми. Правда, персональные компьютеры, построенные на базе микропроцессоров 80286 и 80386, имеют потенциальную возможность работать с 16 Мбайт основной памяти под управлением операционных систем MS OS/2 и XENIX, однако это слабое утешение для миллионов пользователей MS-DOS и компьютеров с микропроцессорами 8086 и 8088.

На весенней компьютерной выставке 1985 г. фирмы Lotus Development Corporation и Intel Corporation совместно объявили о спецификации дополнительной памяти[1] (Expanded Memory Specification, EMS) версии 3.0, разработанной с целью помешать быстрому устареванию выпущенных ранее моделей PC, имеющих ограниченную память. Вскоре после этого фирма Microsoft объявила о своей поддержке EMS и о намерении улучшить интерфейс Microsoft Windows, который в новом варианте будет использовать память, предоставляемую аппаратным и программным обеспечением EMS. Версии EMS 3.2 и 4.0, выпущенные осенью 1985 и летом 1987 г., обеспечили поддержку многозадачных операционных систем.

Дополнительная память LIM EMS (как ее обычно называют) имела огромный успех. Платы памяти EMS выпускаются десятками фирм, и «EMS-ориентированное» программное обеспечение, особенно электронные таблицы, кэш-память для дисков и резидентные утилиты, стало скорее правилом, чем исключением.

Что такое дополнительная память?

Спецификация дополнительной памяти Lotus/Intel/Microsoft представляет функциональное описание подсистемы расширения памяти с переключением банков. Подсистема состоит из аппаратных модулей дополнительной памяти и резидентного драйвера. В версиях EMS 3.0 и 3.2 дополнительная память предоставляется прикладным программам в виде 16-Килобайт страниц, отображаемых на непрерывную 64-Килобайт область, называемую кадром страниц и располагаемую за пределами основной области, используемой MS-DOS/PC-DOS (0–640 Кбайт). Конкретный адрес кадра страниц устанавливается пользователем, что позволяет избежать возможных конфликтов с другой дополнительной аппаратурой. В EMS версии 4.0 страницы могут иметь размер, отличающийся от 16 Кбайт, и отображаться на любые адреса памяти.

EMS предоставляет прикладным программам единообразные средства доступа к 8-Мбайт памяти (в EMS 4.0–32 Мбайт). Поддерживающее программное обеспечение, называемое диспетчером дополнительной памяти (Expanded Memory Manager, EMM), обеспечивает аппаратно-независимый интерфейс между прикладной программой и платой (или платами) дополнительной памяти. Диспетчер EMM поставляется в виде устанавливаемого драйвера устройства, который включается в систему MS-DOS/PC-DOS добавлением строки к файлу CONFIG.SYS на системном загружаемом диске.

Диспетчер дополнительной памяти состоит из двух основных частей, которые можно назвать драйвером и диспетчером. Драйвер имитирует некоторые действия

[1] Используется также термин «отображаемая память», которым подчеркивается метод адресации к EMS – с помощью последовательного отображения отдельных участков этой памяти на относительно узкое окно в адресном пространстве компьютера. – *Прим. перев.*

настоящего устанавливаемого драйвера устройства в том отношении, что он включает функции инициализации и состояния вывода, а также обычный заголовок устройства. Вторая, основная, часть EMM представляет собой интерфейс между прикладным программным обеспечением и аппаратурой дополнительной памяти. Интерфейс предоставляет следующие виды услуг:

- Проверка функционирования аппаратных и программных модулей.
- Выделение страниц дополнительной памяти.
- Отображение логических страниц на кадр физических страниц.
- Освобождение страниц дополнительной памяти.
- Поддержка многозадачных операционных систем.

Прикладные программы взаимодействуют с EMM непосредственно с помощью программного прерывания Int 67H. MS-DOS версий 3.3 и более ранних не принимает

```
        .
        .
        .
                                ; Попытка "открыть" EMM...
        mov     dx,seg emm_name ; DS:DX = адрес имени диспетчера
        mov     ds,dx           ; дополнительной памяти (EMM)
        mov     dx,offset emm_name
        mov     ax,3d00h        ; Функция 3dh, режим = 00h
                                ; = открыть, только для чтения
        int     21h             ; Переход в MS-DOS
        jc      error           ; Переход, если открыть не удалось

                                ; Открыли успешно, убедитесь,
                                ; что это не файл...
        mov     bx,ax           ; BX = полученный дескриптор открытого
                                ; устройства
        mov     ax,4400h        ; Функция 44h, подфункция 00h = IOCTL,
                                ; получить информацию об устройстве
        int     21h             ; Переход в MS-DOS
        jc      error           ; Переход, если вызов IOCTL не прошел
        and     dx,80h          ; Если символьное устройство, бит 7 = 1
        jz      error           ; Переход, если это был файл

                                ; EMM присутствует, убедитесь,
                                ; что он доступен
                                ; (в BX все еще дескриптор)
        mov     ax,4407h        ; Функция 44h, подфункция 07h
                                ; IOCTL, получить состояние вывода
        int     21h             ; Переход в MS-DOS
        jc      error           ; Переход, если вызов IOCTL не прошел
        or      al,al           ; Проверить состояние устройства
        jz      error           ; Если AL = 0, EMM недоступен
                                ; Теперь закроем дескриптор...
                                ; (в BX все еще дескриптор)
        mov     ah,3eh          ; Функция 3eh = закрыть
        int     21h             ; Переход в MS-DOS
        jc      error           ; Переход, если закрыть не удалось
        .
        .
        .
emm_name db     'EMMXXX0',0     ; Гарантированное имя устройства для
                                ; диспетчера дополнительной памяти
```

Рис. 11.4. Проверка наличия диспетчера дополнительной памяти с помощью функций открытия и управления вводом-выводом.

никакого участия в каких бы то ни было обращениях к дополнительной памяти (и фактически их совершенно не замечает). С другой стороны, MS-DOS версии 4.0 и Microsoft Windows являются «EMS-ориентированными» и при наличии дополнительной памяти могут ее использовать.

Дополнительную память не следует путать с расширенной. Термин «расширенная память» используется фирмой IBM по отношению к физическим адресам за пределами 1 Мбайт, к которым имеется доступ в защищенном режиме процессоров 80286 и 80386. Современные версии MS-DOS работают на микропроцессорах 80286 и 80386 в реальном режиме (режиме эмуляции микропроцессора 8086), в результате чего расширенная память прямо не адресуется.

Проверка наличия дополнительной памяти

Для проверки наличия дополнительной памяти прикладная программа может использовать один из двух методов.

Выдать запрос на открытие (Int 21H с функцией 3DH), указав гарантируемое имя устройства для EMM-драйвера EMMXXXX0. Если функция открытия выполнилась успешно, то либо драйвер присутствует, либо на текущем диске случайно имеется файл с таким же именем. Чтобы исключить вторую возможность, прикладная программа может использовать функции управления вводом-выводом (Int 21H с функцией 44H и подфункциями 00H и 07H), убедиться в присутствии EMM. В любом случае прикладная программа должна затем с помощью прерывания Int 21H с функцией 3EH

```
emm_int  equ     67h             ; Программное прерывание диспет-
                                 ;  чера дополнительной памяти (EMM)
          .
          .
          .
                                 ; Сначала получим содержимое
                                 ; вектора прерывания EMM...
         mov     al,emm_int      ; AL = номер прерывания EMM
         mov     ah,35h          ; Функция 35h = получить вектор
         int     21h             ; Переход в MS-DOS
                                 ; Теперь ES:BX = адрес дескриптора

                                 ; Пусть ES:0000 указывает на
                                 ; начало EMM...
         mov     di,10           ; ES:DI = адрес поля имени
                                 ; в заголовке драйвера
                                 ; DS:SI = имя драйвера EMM
         mov     si,seg emm_name
         mov     ds,si
         mov     si,offset emm_name
         mov     cx,8            ; Длина поля имени
         cld
         repz    cmpsb           ; Сравним имена...
         jnz     error           ; Переход, если драйвер отсутствует
          .
          .
          .
emm_name db      'EMMXXXX0'      ; Гарантированное имя для диспетчера
                                 ; дополнительной памяти
```

Рис. 11.5. Проверка наличия диспетчера дополнительной памяти путем анализа поля имени заголовка устройства в программе драйвера.

закрыть дескриптор, образованный функцией открытия, чтобы его можно было использовать для другого файла или устройства.

По адресу, найденному в векторе прерывания Int 67H, проверить заголовок устройства предполагаемого EMM. Обработчики прерываний и драйверы устройств должны использовать этот метод. Если EMM присутствует, поле имени по относительному адресу OAH в заголовке устройства содержит строку EMMXXXX0. Этот способ почти безопасен; кроме того, он устраняет довольно значительные издержки, связанные с использованием функции открытия. Однако эти действия не очень корректны, так как включают в себя анализ ячейки памяти, не принадлежащей прикладной программе.

Описанные методы проверки наличия диспетчера дополнительной памяти проиллюстрированы на рис. 11.4 и 11.5.

ИСПОЛЬЗОВАНИЕ ДОПОЛНИТЕЛЬНОЙ ПАМЯТИ

Удостоверившись, что программа диспетчера памяти присутствует, прикладная программа может обращаться к ней непосредственно, в обход MS-DOS/PC-DOS, с помощью «прерывания пользователя» 67H. Вызов EMM выглядит следующим образом:

```
mov     ah,function     ; AH определяет тип обслуживания
.                       ; Загрузим остальные регистры
.                       ; значениями, специфическими для
.                       ; запрошенного обслуживания
int     67h
```

В общем случае в AH находится номер функции EMM, в AL – номер подфункции (если имеется), в BX – номер страницы (если нужен), а в DX – дескриптор EMM. Регистры DS:SI и ES:DI используются для передачи адресов массивов или буферов. В разд. 4 настоящей книги «Справочник по функциям Lotus/Intel/Microsoft EMS» приведено подробное описание функций дополнительной памяти.

При возврате из функции EMM ноль в регистре AH свидетельствует об успешном выполнении функции; в противном случае регистр содержит код ошибки с установленным старшим битом (рис. 11.6 и 11.7). Другие величины обычно возвращаются в регистр AL и BX или в буфер пользователя.

Прикладная программа, использующая дополнительную память, должна рассматривать эту память как системный ресурс, аналогично файлу или устройству, и для выделения, доступа или освобождения страниц дополнительной памяти применять только документированные средства EMM. Для такой программы можно рекомендовать следующий алгоритм действий.

1. Установить наличие диспетчера дополнительной памяти одним из двух методов, продемонстрированных на рис. 11.4 и 11.5.

2. Удостоверившись в наличии драйвера, проверить его операционное состояние функцией EMS 40H.

3. Проверить номер версии EMM с помощью функции EMS 46H, чтобы убедиться, что средства, требуемые прикладной программой, имеются в системе.

4. Получить сегмент кадра страниц, используемого EMS, с помощью функции 41H.

5. Распределить требуемое число страниц дополнительной памяти с помощью функции EMS 43H. Если распределение выполнено успешно, EMM возвращает дескриптор, который может использоваться прикладной программой для ссылок на страницы дополнительной памяти, которыми владеет программа. Этот шаг в точности аналогичен открытию файла и использованию дескриптора, полученного от функции открытия, для операций чтения-записи файла.

Код ошибки	Значение
00H	Функция выполнена успешно.
80H	Внутренняя ошибка программы диспетчера дополнительной памяти (может быть вызвана разрушением образа драйвера в памяти)
81H	Неправильное функционирование аппаратуры дополнительной памяти
82H	Диспетчер памяти занят
83H	Недопустимый дескриптор
84H	Функция, затребованная прикладной программой, не определена
85H	Нет свободных дескрипторов
86H	Ошибка при сохранении или восстановлении контекста отображения
87H	В запросе на выделение указано больше логических страниц, чем физически имеется в системе; выделение страниц не выполнялось
88H	В запросе на выделение указано больше логических страниц, чем число свободных страниц в настоящий момент (запрошенное число не превышает числа существующих физических страниц, но некоторые страницы уже выделены другим дескриптором); выделение страниц не выполнялось
89H	Нулевые страницы; не подлежат выделению
8AH	Логическая страница, для которой запрошено отображение, расположена вне диапазона логических страниц, назначенных дескриптору
8BH	В запросе на отображение указан неверный номер физической страницы (не в пределах 0–3)
8CH	Область сохранения отображения страниц и состояния аппаратуры заполнена
8DH	Не выполнено сохранение контекста отображения; область сохранения уже содержит контекст, соответствующий затребованному дескриптору
8EH	Не выполнено восстановление контекста отображения; область сохранения не содержит контекст для затребованного дескриптора
8FH	Не определен параметр подфункции

Рис. 11.6. Коды ошибок диспетчера дополнительной памяти, общие для версий EMS 3.0, 3.2 и 4.0.

После обращения к EMM регистр AH содержит 0, если функция выполнилась успешно и код ошибки в диапазоне от 80H до 8FH, если функция не выполнилась.

Код ошибки	Значение
90H	Не определен тип атрибута
91H	Средство не поддерживается
92H	Районы памяти источника и приемника имеют один и тот же дескриптор и перекрываются; затребованная пересылка выполнена, но часть района источника заполнена новыми данными
93H	Указанная длина района памяти источника или приемника больше фактически назначенной длины
94H	Район обычной памяти и район дополнительной памяти перекрываются
95H	Указанный относительный адрес выходит за пределы логической страницы
96H	Длина района превышает 1 Мбайт
97H	Районы памяти источника и приемника имеют один и тот же дескриптор и перекрываются; обмен не может быть выполнен.
98H	Не определены типы памяти источника и приемника
99H	Этот код ошибки в настоящее время не используется
9AH	Дополнительные наборы регистров отображения или ПДП поддерживаются, но указанный дополнительный набор регистров не поддерживается
9BH	Дополнительные наборы регистров отображения или ПДП поддерживаются, но все дополнительные наборы регистров уже выделены
9CH	Дополнительные наборы регистров отображения или ПДП не поддерживаются, и указан не нулевой дополнительный набор регистров

Рис. 11.7. Коды ошибок диспетчера дополнительной памяти, используемые только в EMS версии 4.0.

Большинство этих ошибок относится к функциям EMS, предназначенным для использования операционными системами и обычно не встречающимся в прикладных программах.

9DH	Дополнительные наборы регистров отображения или ПДП поддерживаются, но указанный дополнительный набор регистров не определен или не выделен
9EH	Выделенные каналы ПДП не поддерживаются
9FH	Выделенные каналы ПДП поддерживаются, но указанный канал ПДП не поддерживается
A0H	Для указанного имени не найден дескриптор
A1H	Дескриптор с таким именем уже существует
A2H	Оборачивание адреса памяти; сумма базового адреса района источника или приемника и длины превышает 1 Мбайт
A3H	Функции передан недопустимый указатель или содержимое массива-источника разрушено
A4H	Доступ к функции запрещен операционной системой

Рис. 11.7.

6. Если запрошенного числа страниц нет в наличии, прикладная программа может запросить у EMM информацию о фактическом числе наличных страниц (функция EMS 42H) и определить целесообразность продолжения работы.

7. После того как прикладная программа успешно выделила требуемое число страниц дополнительной памяти, она использует функцию EMS 44H для отображения (или снятия отображения) логических страниц на кадр физических страниц с целью записи и извлечения данных из дополнительной памяти.

8. Программа, закончив работу со страницами дополнительной памяти, должна с помощью функции EMS 45H освободить их. В противном случае страницы будут потеряны в смысле невозможности их использования другими программами до перезапуска системы.

На рис. 11.8 показана структура программы, использующая этот общий алгоритм.

```
.
.
.
mov     ah,40h          ; Проверка состояния EMM
int     67h
or      ah,ah
jnz     error           ; Переход, если состояние EMM
                        ; не нормально
mov     ah,46h          ; Проверка версии EMM
int     67h
or      ah,ah
jnz     error           ; Переход, если не смогли
                        ; получить версию

cmp     al,030h         ; Убедимся, что по меньшей мере версия 3.0
jb      error           ; Переход, если неверная версия EMM
mov     ah,41h          ; Получить сегмент кадра страниц
int     67h
or      ah,ah
jnz     error           ; Переход, если кадр не получен
mov     page_frame,bx   ; Сохранить сегмент кадра страниц

mov     ah,42h          ; Получить число доступных страниц
int     67h
or      ah,ah
jnz     error           ; Переход, если ошибка получения страниц
```

Рис. 11.8. Программа, иллюстрирующая общий алгоритм использования дополнительной памяти.

```
mov     total_pages,dx  ; Сохранить полное число страниц EMM
mov     avail_pages,bx  ; Сохранить доступное число страниц EMM
or      bx,bx
jz      error           ; Аварийное завершение, если нет страниц

mov     ah,43h          ; Попытка выделить страницы EMM
mov     bx,needed_pages
int     67h             ; Если выделение  прошло успешно
or      ah,ah
jnz     error           ; Переход, если ошибка выделения

mov     emm_handle,dx   ; Сохранить дескриптор
                        ; выделенных страниц

.
.                       ; Теперь мы готовы к любым операциям
.                       ; с использованием страниц EMM
.

                        ; Отображение страницы памяти EMS...
mov     bx,log_page     ; BX <- номер логической страницы EMS
mov     al,phys_page    ; AL <- физическая страница EMS (0-3)
mov     dx,emm_handle   ; Дескриптор EMM для наших страниц
mov     ah,44h          ; Функция 44h = отобразить страницу EMS
int     67h
or      ah,ah
jnz     error           ; Переход, если ошибка отображения

.
.
.                       ; Программа готова к завершению,
                        ; отдадим выделенные страницы EMM...
mov     dx,emm_handle   ; Дескриптор наших страниц
mov     ah,45h          ; Функция 45h EMS = освободить страницы
int     67h
or      ah,ah
jnz     error           ; Переход, если освобождение не прошло
.
.
.
```

Рис. 11.8.

Обработчики прерываний или драйверы устройств, использующие EMS, следуют общей процедуре, описанной выше (пункты 1–8), с несколькими незначительными отклонениями. Этим программам может понадобиться получить дескриптор EMS и выделить страницы еще перед тем, как операционная система стала функционировать полностью; в частности, могут оказаться недоступными функции MS-DOS открытия файла или устройства, управления вводом-выводом и получения вектора прерывания. Такие обработчики прерываний или драйверы должны для проверки наличия EMM использовать модифицированный вариант методики «получить вектор прерываний» (см. рис. 11.5), непосредственно извлекая содержимое вектора прерывания Int 67H.

Драйверы устройств и обработчики прерываний обычно захватывают выделенную им дополнительную память и не освобождают ее до перезапуска системы. Такие программы должны также позаботиться о сохранении и восстановлении контекста отображения страниц EMM (функции EMS 47H и 48H) при каждом обращении

к дополнительной памяти, чтобы не нарушить использование EMS фоновой программой.

Прикладные программы должны корректно работать с EMM и не допускать разрушения дополнительной памяти. Если несколько прикладных программ выполняются под управлением многозадачного администратора, например Microsoft Windows, и хотя бы одна из этих программ отходит от строгого выполнения соглашений EMM, данные некоторых или всех прикладных программ могут оказаться уничтоженными.

РАСШИРЕННАЯ ПАМЯТЬ

Расширенной памятью называется часть основной памяти с произвольной выборкой с адресами, превышающими 1 Мбайт (100000H), которая доступна процессорам 80286 и 80386, работающим в защищенном режиме. Машины, совместимые с IBM PC/AT и PS/2, могут (теоретически) иметь до 15 Мбайт установленной расширенной памяти в дополнение к обычной памяти объемом 1 Мбайт.

Операционные системы защищенного режима, такие, как Microsoft XENIX или MS OS/2, могут использовать расширенную память для выполнения программ. С другой стороны, MS-DOS работает на процессорах 80286 и 80386 в реальном режиме, и программы, выполняемые под ее управлением, обычно не могут располагаться в расширенной памяти или даже адресоваться к ней с целью сохранения данных. Однако ПЗУ BIOS включает две программы, которые позволяют программам реального режима получить ограниченный доступ к расширенной памяти.

Функции ПЗУ BIOS	Действие
Int 15H, функция 87H	Переслать блок расширенной памяти
Int 15H, функция 88H	Получить размер расширенной памяти

С помощью этих функций электронные диски и другие программы могут использовать расширенную память для быстрого сохранения и извлечения информации, которая в противном случае должна была бы записываться на более медленный физический диск. В разд. 3 этой книги «Справочник по функциям ПЗУ BIOS и манипулятора мышь» приведено описание обеих этих функций.

Следует с осторожностью использовать эти программы ПЗУ BIOS. Расширенная память, разумеется, энергозависима; записанные в нее данные теряются при выключении машины. Передача данных в расширенную память и из нее требует переключения из реального режима в защищенный и обратно – процесс довольно медленный на машинах с процессором 80286; в некоторых случаях он лишь незначительно быстрее чтения данных с жесткого диска. К тому же программы, использующие функции расширенной памяти ПЗУ BIOS, не могут работать в режиме совместимости с MS-DOS системы MS OS/2.

Наконец, основной недостаток этих функций BIOS заключается в том, что они не делают никаких попыток арбитража двух или более программ или драйверов, использующих расширенную память для временного хранения информации. Если, например, прикладная программа и установленный драйвер электронного диска пытаются поместить данные в одну и ту же область расширенной памяти, ни в одну из программ не вернется состояние ошибки, хотя данные одной или обеих программ могут быть уничтожены.

На рис. 11.9 приведен пример программных строк, необходимых для пересылки данных в расширенную память и из нее.

```
bmdt    db      30h dup (0)     ; Таблица дескрипторов перемещения блоков

buff1   db      80h dup ('?')   ; Буфер источника
buff2   db      80h dup (0)     ; Буфер приемника

        .
        .
        .

                                ; Скопируем 'buff1' по адресу
                                ; 100000h расширенной памяти
        mov     dx,10h          ; DX:AX = адрес приемника
        mov     ax,0            ; в расширенной памяти
        mov     bx,seg buff1    ; DX:BX = адрес источника
        mov     ds,bx           ; в обычной памяти
        mov     bx,offset buff1
        mov     cx,80h          ; CX = число пересылаемых байтов
        mov     si,seg bmdt     ; ES:SI = переслать блок
        mov     es,si           ; Таблица дескрипторов
        mov     si,offset bmdt
        call    putblk          ; Запрос на пересылку

                                ; Заполним buff2 из расширенной
                                ; памяти (адрес 100000h)
        mov     dx,10h          ; DX:AX = адрес источника
        mov     ax,0            ; в расширенной памяти
        mov     bx,seg buff2    ; DS:BX = адрес приемника
        mov     ds,bx           ; в обычной памяти
        mov     bx,offset buff2
        mov     cx,80h          ; CX = число пересылаемых байтов
        mov     si,seg bmdt     ; ES:SI = переслать блок
        mov     es,si           ; Таблица дескрипторов
        mov     si,offset bmdt
        call    getblk          ; Запрос на пересылку

        .
        .
        .

getblk  proc    near            ; Пересылка блока из расширенной памяти
                                ; в реальную
                                ; При вызове:
                                ; DX:AX = линейный 32-разрядный адрес
                                ;         источника в расширенной памяти
                                ; DS:BX = адрес  приемника -
                                ;         сегмент и относительный адрес
                                ; CX    = длина в байтах
                                ; ES:SI = дескриптор таблицы
                                ;         пересылки блока
                                ; При возврате:
                                ; AH = 0, если пересылка выполнена успешно

        mov     es:[si+10h],cx  ; Сохраним длину в дескрипторах
```

Рис. 11.9. Пересылка блоков данных между обычной и расширенной памятью с помощью функций расширенной памяти ПЗУ BIOS.

Дополнительную информацию о формате таблицы дескрипторов пересылки блока можно найти в описании прерывания Int 15H с функцией 87H в разд. 3 этой книги «Справочник по функциям ПЗУ BIOS и манипулятора мышь». Обратите внимание на необходимость указания адреса расширенной памяти не в виде сегмента и относительного адреса, а как 32-разрядного линейного адреса.

```
        mov     es:[si+18h],cx

                                ; Сохраним байты прав доступа
        mov     byte ptr es:[si+15h],93h
        mov     byte ptr es:[si+1dh],93h

        mov     es:[si+12h],ax  ; Адрес источника в расширенной памяти
        mov     es:[si+14h],dl

                                ; Преобразуем сегмент и относительный
                                ; адрес приемника в линейный адрес
        mov     ax,ds           ; сегмента * 16
        mov     dx,16
        mul     dx
        add     ax,bx           ; + относительный адрес -> линейный адрес
        adc     dx,0

        mov     es:[si+1ah],ax  ; Сохраним адрес приемника
        mov     es:[si+1ch],dl

        shr     cx,1            ; Преобразуем длину в слова
        mov     ah,87h          ; Int 15h, функция 87h = пересылка блока
        int     15h             ; Переход в BIOS

        ret                     ; Возврат в запрашивающую программу

getblk  endp

putblk  proc    near            ; Пересылка блока из реальной памяти
                                ; в расширенную
                                ; При вызове:
                                ; DX:AX = линейный 32-разрядный адрес
                                ;         приемника в расширенной памяти
                                ; DS:BX = адрес источника -
                                ;         сегмент и относительный адрес
                                ; CX     = длина в байтах
                                ; ES:SI = дескриптор таблицы
                                ;         пересылки блока
                                ; При возврате:
                                ; AH = 0, если пересылка выполнена успешно

        mov     es:[si+10h],cx  ; Сохраним длину в дескрипторах
        mov     es:[si+18h],cx

                                ; Сохраним байты прав доступа
        mov     byte ptr es:[si+15h],93h
        mov     byte ptr es:[si+1dh],93h

        mov     es:[si+1ah],ax  ; Сохраним  адрес приемника
        mov     es:[si+1ch],dl  ; в расширенной памяти

                                ; Преобразуем сегмент и относительный
                                ; адрес источника в линейный адрес
        mov     ax,ds           ; сегмента * 16
        mov     dx,16
        mul     dx
        add     ax,bx           ; + относительный адрес -> линейный адрес
        adc     dx,0
```

Рис. 11.9.

```
        mov     es:[si+12h],ax   ; Сохраним адрес источника
        mov     es:[si+14h],dl

        shr     cx,1             ; Преобразуем длину в слова
        mov     ah,87h           ; Int 15h, функция 87h = пересылка блока
        int     15h              ; Переход в BIOS

        ret                      ; Возврат в запрашивающую программу

putblk endp
```

Рис. 11.9.

ГЛАВА 12. ФУНКЦИЯ EXEC

Функция MS-DOS EXEC (Int 21H, функция 4BH) позволяет программе (называемой родительской) загрузить любую другую программу (называемую дочерней), находящуюся во внешней памяти, выполнить ее, а после завершения дочерней программы продолжить свое выполнение.

Родительская программа может передавать дочерней информацию в командной строке, в блоках управления файлами или с помощью набора строк, называемого блоком окружения (будет описан ниже). Все файлы или устройства, открытые родительской программой с помощью функций управления файлами типа дескриптора, дублируются в созданной заново дочерней задаче; другими словами, дочерняя задача наследует все активные дескрипторы родительской. Все файловые операции над дескрипторами, выполняемые дочерней задачей, например поиск или ввод-вывод, отражаются на состоянии дескрипторов файлов родительской задачи.

MS-DOS приостанавливает выполнение родительской программы до завершения дочерней. Дочерняя программа, закончив выполнение, может передать родительской код завершения с информацией о встретившихся ошибках. Она может в свою очередь загружать другие программы, а те – следующие, передавая управление через много уровней, пока в системе не исчерпается память.

Интерпретатор команд MS-DOS COMMAND.COM использует функцию EXEC для запуска своих внешних команд и других прикладных программ. Многие популярные коммерческие программы, такие, как администраторы баз данных и текстовые процессоры, с помощью EXEC запускают другие программы (например, орфографический корректор) или загружают вторую копию COMMAND.COM, чтобы дать возможность пользователю распечатывать каталоги и копировать или переименовывать файлы, не закрывая для этого все прикладные файлы и не останавливая основной работы. Функцию EXEC можно использовать и для загрузки программных оверлейных сегментов, хотя это необычная область применения.

ПОЛУЧЕНИЕ ПАМЯТИ

Для того чтобы родительская программа могла с помощью функции EXEC загрузить дочернюю, в транзитной области должен иметься достаточный объем нераспределенной памяти.

Когда загружается сама родительская программа, MS-DOS выделяет ей тот или иной объем памяти исходя из типа файла с программой – .COM. или .EXE и другой информации, доступной загрузчику. (Подробности приведены в гл. 11.) Из-за того что операционная система не имеет надежного способа предугадать, сколько памяти потребуется той или иной программе, она обычно выделяет программе гораздо больше памяти, чем ей в действительности необходимо. Поэтому первым действием

будущего «родителя» должен быть вызов Int 21H с функцией 4AH (Изменить размер блока памяти), чтобы вернуть MS-DOS всю лишнюю собственную память. В этом случае при вызове Int 21N с функцией 4AH регистр ES должен указывать на префикс программного сегмента программы, освобождающей память, а регистр BX – содержать число параграфов памяти, которые необходимо оставить за программой (см., например, рис. 11.1).

Предупреждение. Программа .COM должна сохранить свой стек в надежном месте, если она уменьшает объем распределенной ей памяти до величины менее 64 Кбайт.

ЗАПРОС ФУНКЦИИ EXEC

Для загрузки и выполнения дочерней программы родительская программа должна выполнить Int 21H при следующем состоянии регистров:

AH = 4BH
AL = 00H (подфункция загрузки дочерней программы)
DS:DX = сегмент:относительный_адрес пути к дочерней программе
ES:BX = сегмент:относительный_адрес блока параметров

Блок параметров в свою очередь содержит адреса остальной информации, требующейся функции EXEC.

Имя программы

Имя запускаемой программы, которое вызывающая программа передает функции EXEC, должно представлять собой однозначную спецификацию файла (без символов групповых операций), в явной форме включающую расширение .COM или .EXE. Если в передаваемое имя программы не входят путь и обозначение дисковода, MS-DOS использует текущий каталог и подразумеваемый дисковод. (Последовательный поиск файлов .COM, .EXE и .BAT во всех каталогах, перечисленных в переменной PATH, не входит в функции программы EXEC, а является внутренней задачей COMMAND.COM.)

Командный файл .BAT нельзя запустить с помощью функции EXEC непосредственно; вместо этого следует, используя EXEC, вызвать копию COMMAND.COM и передать имя командного файла в хвосте команды после ключа /C.

Блок параметров

Блок параметров включает адреса четырех объектов:

- Блока окружения
- Хвоста команды
- Двух используемых по умолчанию блоков управления файлами

Место, зарезервированное в блоке параметров под адрес блока окружения, составляет всего два байта и содержит адрес сегмента. Оставшиеся три адреса являются двухсловными; каждый из них занимает 4 байт, причем в первых двух байтах хранится относительный адрес, а во вторых двух – адрес сегмента.

Блок окружения

Каждая программа, загружаемая с помощью функции EXEC, наследует от своего родителя структуру данных, называемую блоком окружения (environment block).

Указатель на сегмент блока находится по относительному адресу 002CH в префиксе программного сегмента (PSP). В блоке окружения хранится определенная информация. используемая системным интерпретатором команд (обычно COMMAND.COM), и, кроме того, может содержаться информация, используемая транзитными программами. Этот блок не оказывает никакого влияния на функционирование самой операционной системы.

Если указатель блока окружения в блоке параметров функции EXEC содержит 0, дочерняя программа получает копию блока окружения родительской программы. В противном случае дочерней программе передается указатель сегмента другого или расширенного окружения. Максимальный размер блока окружения составляет 32 Кбайт, так что с помощью этого механизма от программы к программе могут передаваться весьма значительные объемы информации.

Блок окружения любой конкретной программы является статическим в том смысле, что если в основной памяти находятся несколько поколений дочерних программ, каждая будет иметь свою собственную отдельную копию блока окружения. Более того, блок окружения программы, которая после завершения остается резидентной, не модифицируется последующими командами PATH и SET.

Позже в этой главе будут приведены дополнительные сведения о блоке окружения.

Хвост команды

MS-DOS копирует хвост команды в PSP дочерней программы по относительному адресу 0080H, как это было описано в гл. 3. Информация имеет форму байта счетчика, за которым следует строка символов ASCII, завершаемая возвратом каретки; возврат каретки не включается в число в счетчике.

Хвост команды может включать имена файлов, ключи и другие параметры. С точки зрения дочерней программы хвост команды должен содержать ту же информацию, которая присутствовала бы при непосредственном запуске программы командой пользователя по запросу MS-DOS. Функция EXEC игнорирует любые параметры перенаправления ввода-вывода, помещенные в хвост команды; родительская программа должна организовать перенаправление стандартных устройств перед вызовом EXEC.

Блоки управления файлами, используемые по умолчанию

MS-DOS копирует два используемых по умолчанию блока управления файлами, на которые указывает блок параметров функции EXEC, в PSP дочерней программы по относительным адресам 005CH и 006CH. Для эмуляции функций COMMAND.COM с точки зрения дочерней программы родительская программа должна использовать прерывание Int 21H с функцией 29H (системный анализ имени файла) для извлечения двух первых параметров хвоста команды в используемые по умолчанию блоки управления файлами перед вызовом функции EXEC.

От блоков управления файлами мало проку для MS-DOS версий 2 и 3, потому что они не поддерживают иерархическую файловую структуру, но некоторые прикладные программы обращаются к ним, чтобы быстро получить два первых ключа или другие параметры из хвоста команды. В гл. 8 блоки управления файлами обсуждаются подробнее.

ВОЗВРАТ ИЗ ФУНКЦИИ EXEC

В MS-DOS версии 2 функция EXEC уничтожает сдержимое всех регистров, за исключением регистра сегмента программы (CS) и указателя команд (IP). Поэтому перед вызовом EXEC родительская программа должна поместить содержимое любых регистров, в которых есть что-то важное, в стек, а затем сохранить в переменных регистр сегмента стека (SS) и указатель стека (SP). После возврата (при условии успешной отработки вызова EXEC, т.е. завершения дочерней программы) родительская программа должна перезагрузить SS и SP из переменных, где они были сохранены, а затем извлечь из стека содержимое остальных сохраненных там регистров. В MS-DOS версий 3.0 и выше указатель стека и другие регистры автоматически сохраняются при вызове EXEC и восстанавливаются при возврате обычным образом.

Наконец, родительская программа может использовать прерывание Int 21H с функцией 4DH для получения типа завершения и кода возврата из дочерней программы.

Функция EXEC не будет выполнена при следующих обстоятельствах:

- Для загрузки и выполнения затребованного программного файла не хватает невыделенной памяти
- Запрошенная программа не найдена на диске
- Транзитная часть COMMAND.COM в старших адресах ОЗУ (содержащая собственно загрузчик) разрушена и свободной памяти не хватает для ее перезагрузки (только PC-DOS версии 2).

На рис. 12.1 собраны характеристики вызова 4BH. На рис. 12.2 показана структура типичного вызова EXEC. В этом конкретном примере функция EXEC использована для загрузки и запуска утилиты MS-DOS CHKDSK.COM. Листинг программы

При вызове
AH = 4BH
AL = тип функции
00 = загрузка и выполнение программы
03 = загрузка оверлея
ES:BX = сегмент:относительный_адрес блока параметров
DS:DX = сегмент:относительный_адрес спецификации программы

При возврате
В случае успешного выполнения
Флаг переноса сброшен. В MS-DOS версии 2 все регистры, кроме CS:IP, могут быть разрушены. В MS-DOS версий 3.0 и выше регистры сохраняются обычным образом
В случае невыполнения
Флаг переноса установлен и AX = код ошибки

Формат блока параметров
Если AL = 0 (загрузка и выполнение программы)
Байты 0–1 = указатель сегмента, блок окружения
Байты 2–3 = относительный адрес хвоста командной строки
Байты 4–5 = сегмент хвоста командной строки
Байты 6–7 = относительный адрес первого блока управления файлами, копируемого в новый PSP + 5CH
Байты 8–9 = сегмент первого блока управления файлами
Байты 10–11 = относительный адрес второго блока управления файлами, копируемого в новый PSP + 6CH.
Байты 12–13 = сегмент второго блока управления файлами
Если AL = 3 (загрузка оверлея)
Байты 0–1 = адрес сегмента, куда будет загружен файл
Байты 2–3 = фактор перемещения для загруженного образа

Рис. 12.1. Характеристики вызова функции EXEC (прерывание Int 21H с функцией 4BH).

```
cr      equ     0dh             ; Код ASCII возврата каретки
        .
        .
        .
        mov     stkseg,ss       ; Сохраним указатель стека
        mov     stkptr,sp

        mov     dx,offset pname ; DS:DX = имя программы
        mov     bx,offset pars  ; ES:BX = блок параметров
        mov     ax,4b00h        ; Функция 4bh, подфункция 00h
        int     21h             ; Переход в MS-DOS

        mov     ax,_DATA        ; Сделаем наш сегмент данных
        mov     ds,ax           ; снова адресуемым
        mov     es,ax

        cli                     ; (Ввиду ошибки в некоторых МП 8088)
        mov     ss,stkseg       ; Восстановить указатель стека
        mov     sp,stkptr
        sti                     ; (Ввиду ошибки в некоторых МП 8088)

        jc      error           ; Переход, если EXEC не прошла
        .
        .
        .

stkseg  dw      0               ; Первоначальное содержимое SS
stkptr  dw      0               ; Первоначальное содержимое SP

pname   db      '\CHKDSK.COM',0 ; Путь к дочерней программе

pars    dw      envir           ; Сегмент контекста
        dd      cmdline         ; Командная строка для дочерней программы
        dd      fcb1            ; Блок управления файлом #1
        dd      fcb2            ; Блок управления файлом #2

cmdline db      4,' *.*',cr     ; Командная строка для дочерней програмы

fcb1    db      0               ; Блок управления файлом #1
        db      11 dup ('?')
        db      25 dup (0)

fcb2    db      0               ; Блок управления файлом #2
        db      11 dup (' ')
        db      25 dup (0)

envir   segment para 'ENVIR'    ; Сегмент контекста

        db      'PATH=',0       ; Пустой путь поиска
                                ; Местоположение файла COMMAND.COM
        db      'COMSPEC=A:\COMMAND.COM',0
        db      0               ; Конец контекста

envir   ends
```

Рис. 12.2. Короткий пример использования вызова MS-DOS EXEC со всеми необходимыми переменными и управляющими блоками.

Обратите внимание на защиту регистров для версии 2 MS-DOS и маскирование прерываний на время загрузки SS:SP, позволяющее избежать проявления ошибки, имевшейся в некоторых ранних процессорах 8088.

SHELL.ASM, приведенный далее в этой главе (рис. 12.5), содержит более полный пример использования прерывания Int 21H с функцией 4AH для освобождения ненужной памяти.

ДОПОЛНИТЕЛЬНЫЕ СВЕДЕНИЯ О БЛОКЕ ОКРУЖЕНИЯ

Блок окружения всегда выравнивается на параграф (начинается по адресу, кратному 16 байт) и содержит ряд строк ASCIIZ, имеющих форму

имя = параметр

Дополнительный нулевой байт (рис. 12.3) указывает на конец всей последовательности строк. В MS-DOS версии 3 за блоком строк окружения и дополнительным нулевым байтом размещаются счетчик слов и полная спецификация файла (дисковод, путь, имя файла и расширение), используемая функцией EXEC для загрузки программы.

В обычном случае блок окружения, наследуемый программой, содержит по меньшей мере три строки:

COMSPEC = переменная
PATH = переменная
PROMPT = переменная

MS-DOS помещает эти три строки в блок окружения при инициализации системы, в процессе интерпретации директив SHELL, PATH и PROMPT в файлах CONFIG. SYS и AUTOEXEC.BAT. Строки говорят интерпретатору команд MS-DOS COMMAND. COM, где находится его выполнимый файл (чтобы COMMAND.COM мог загрузить свою транзитную часть), где искать внешние команды или выполнимые программные файлы, а также каков формат запроса пользователя на экране терминала.

Вы можете добавить в блок окружения дополнительные строки либо в интерактивном режиме, либо в виде командных файлов с помощью команды SET. Транзитные программы могут использовать эти строки с целью передачи информации. Например, транслятор с языка Си Microsoft C Compiler просматривает содержимое блока окружения в поисках строк INCLUDE, LIB и TMP, в которых указано, где искать файлы include и библиотечные файлы и где располагать временные рабочие файлы.

```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000  43 4F 4D 53 50 45 43 3D 43 3A 5C 43 4F 4D 4D 41  COMSPEC=C:\COMMA
0010  4E 44 2E 43 4F 4D 00 50 52 4F 4D 50 54 3D 24 70  ND.COM.PROMPT=$p
0020  24 5F 24 64 20 20 20 24 74 24 68 24 68 24 68 24  $_$d   $t$h$h$h$
0030  68 24 68 24 68 20 24 71 24 71 24 67 00 50 41 54  h$h$h $q$q$g.PAT
0040  48 3D 43 3A 5C 53 59 53 54 45 4D 3B 43 3A 5C 41  H=C:\SYSTEM;C:\A
0050  53 4D 3B 43 3A 5C 57 53 3B 43 3A 5C 45 54 48 45  SM;C:\WS;C:\ETHE
0060  52 4E 45 54 3B 43 3A 5C 46 4F 52 54 48 5C 50 43  RNET;C:\FORTH\PC
0070  33 31 3B 00 00 01 00 43 3A 5C 46 4F 52 54 48 5C  31;....C:\FORTH\
0080  50 43 33 31 5C 46 4F 52 54 48 2E 43 4F 4D 00 20  PC31\FORTH.COM.
```

Рис. 12.3. Дамп типичного блока окружения версии 3 MS-DOS.

Этот конкретный пример содержит используемый по умолчанию параметр COMSPEC и две относительно сложные управляющие строки PATH и PROMPT, установленные с помощью файла пользователя AUTOEXEC. Обратите внимание на описание пути и спецификацию файла выполняемой программы, расположенные вслед за двойным нулем по относительному адресу 0073H. Двойной нуль отмечает конец блока окружения.

ПРИМЕРЫ ПРОГРАММ: SHELL.C И SHELL.ASM

В качестве практического примера использования функции MS-DOS EXEC приведен небольшой интерпретатор команд по имени SHELL в виде эквивалентных текстов на языке Си (Microsoft C, рис. 12.4) и языке макроассемблера (Microsoft Macro Assembler, рис. 12.5). Исходный текст программы на языке ассемблера значительно сложнее варианта программы на языке Си, но имена и функции их процедур совершенно аналогичны.

```
/*
    SHELL.C     Простой расширяемый интерпретатор команд
                для MS-DOS версий 2.0 и выше
    Издательские права 1988 Рей Данкан
    Компиляция:         C>CL SHELL.C
    Использование:      C>SHELL
*/

#include <stdio.h>
#include <process.h>
#include <stdlib.h>
#include <signal.h>

                                /* Макрокоманда ,возвращающая
                                   число элементов в структуре     */
#define dim(x) (sizeof(x) / sizeof(x[0]))

unsigned intrinsic(char *);     /* Прототипы функций                */
void extrinsic(char *);
void get_cmd(char *);
void get_comspec(char *);
void break_handler(void);
void cls_cmd(void);
void dos_cmd(void);
void exit_cmd(void);

struct cmd_table {              /* Таблица внутренних команд        */
                  char *cmd_name;
                  int  (*cmd_fxn)();
                } commands[] =

                { "CLS",   cls_cmd,
                  "DOS",   dos_cmd,
                  "EXIT",  exit_cmd,  };

static char com_spec[64];       /* Спецификация файла COMMAND.COM   */

main(int argc, char *argv[])
{
   char inp_buf[80];            /* Входной буфер клавиатуры         */

   get_comspec(com_spec);       /* Получить спецификацию файла
                                   COMMAND.COM                      */

                                /* Зарегистрируем новый обработчик
                                   прерывания по Ctrl-C             */
   if(signal(SIGINT, break_handler) == (int(*)()) - 1)
```

Рис. 12.4. SHELL.C. Табличный интерпретатор команд, написанный на языке Microsoft C.

```
    {
        fputs("Не могу отловить прерывание по Control-C",stderr);
        exit(1);
    }

    while(1)                          /* Главный цикл интерпретатора      */
    {
        get_cmd(inp_buf);             /* Получить команду                 */
        if (! intrinsic(inp_buf) )    /* Если команда внутренняя,
                                         запустим ее подпрограмму         */
           extrinsic(inp_buf);        /* В противном случае передача
                                         управления в COMMAND.COM         */
        }
}

/*
   Попробуем найти команду пользователя в таблице внутренних
   команд. Если команда найдена, запустим соответствующую
   подпрограмму и возвратим "истину"; в противном случае
   возвратим "ложь".
*/
unsigned intrinsic(char *input_string)
{
    int i, j;                         /* Временные переменные             */

                                      /* Уберем лидирующие пробелы        */
    while(*input_string == '\x20') input_string++ ;

                                      /* Поиск в таблице команд           */
    for(i=0; i < dim(commands); i++)
    {
        j = strcmp(commands[i].cmd_name, input_string);

        if(j == 0)                    /* Если найдена,
                                         запускаем подпрограмму           */
        {
            (*commands[i].cmd_fxn)();
            return(1);                /* и возвращаем "истину"            */
        }
    }
    return(0);                        /* Не найдена, возвращаем "ложь"    */
}

/*
   Обработка внешней команды путем ее передачи
   копии COMMAND.COM (полученной с помощью функции EXEC)
*/

void extrinsic(char *input_sting)
{
    int status;
    status == system(input_string); /* Вызов функции EXEC                 */

    if(status)                        /* Если не получилось, вывод
                                         сообщения об ошибке              */
        fputs("\nОшибка вызова COMMAND.COM\n", stderr);
}
```

Рис. 12.4.

```
/*
   Вывод запроса, получение команды пользователя со стандартного
   устройства ввода, преобразование в символы верхнего регистра.
*/

void get_cmd(char *buffer)
{
   printf("\nsh: ");              /* Запрос                          */
   gets(buffer);                 /* Получить введенное с клавиатуры */
   strupr(buffer);               /* Преобразовать в символы
                                    верхнего регистра               */
}

/*
   Получение полного пути и спецификации файла COMMAND.COM
   из контекстной переменной COMSPEC
*/
void get_comspec(char *buffer)
{
   strcpy(buffer, getenv("COMSPEC"));

   if(buffer[0] == NULL)
   {
       fputs("\nВ контексте нет COMSPEC\n", stderr);
       exit(1);
   }
}

/*
   Этот обработчик Ctrl-C удерживает SHELL от потери управления.
   Он просто выводит заново запрос и завершается.
*/
void break_handler(void)
{
    signal(SIGINT, break_handler);/* Сброс обработчика              */
    printf("\nsh: ");             /* Запрос, выводимый на экран     */
}

/*
   Это подпрограммы для внутренних команд.
*/

void cls_cmd(void)               /* Команда CLS                     */
{
    printf("\033[2j");           /* Esc-последовательность ANSI     */
}                                /* для очистки экрана              */

void dos_cmd(void)               /* Команда DOS                     */
{
    int status;
                                 /* Запуск COMMAND.COM              */
    status = spawnlp(P_WAIT, com_spec, com_spec, NULL);

    if (status)
        fputs("\nОшибка вызова COMMAND.COM\n",stderr);
}
```

Рис. 12.4.

```
void exit_cmd(void)                /* Команда EXIT                    */
{
    exit(0);                       /* Завершить SHELL                 */
}
```

Рис. 12.4.

```
        name    shell
        page    55,132
        title   SHELL.ASM--simple MS-DOS shell ; Простая оболочка MS-DOS
; SHELL.ASM     Простой расширяемый интерпретатор команд
;               для MS-DOS версий 2.0 и выше
;
; Авторские права 1988 Рей Данкан

; Компоновка:    C>MASM SHELL;
;                C>LINK SHELL;
;
; Использование: C>SHELL;
;

stdin   equ     0               ; Дескриптор стандартного ввода
stdout  equ     1               ; Дескриптор стандартного вывода
stderr  equ     2               ; Дескриптор стандартной ошибки

cr      equ     0dh             ; Символ ASCII возврата каретки
lf      equ     0ah             ; Символ ASCII перевода строки
blank   equ     20h             ; Символ ASCII пробела
escape  equ     01bh            ; Код ASCII символа Escape

_TEXT   segment word public 'CODE'

        assume  cs:_TEXT,ds:_DATA,ss:STACK

shell   proc    far             ; При входе DS = ES = PSP

        mov     ax,_DATA        ; Сделаем наш сегмент данных
        mov     ds,ax           ; адресуемым

        mov     ax,es:[002ch]   ; Получить сегмент контекста
        mov     env_seg,ax      ; из PSP и сохранить его

                                ; Освободить ненужную память...
                                ; ES уже = сегмент PSP
        mov     bx,100h         ; BX = число требуемых параграфов
        mov     ah,4ah          ; Функция 4ah = изменить размер блока памяти
        int     21h             ; Переход в MS-DOS
        jnc     shell1          ; Переход, если изменение размера
                                ; выполнено успешно

        mov     dx,offset msg1  ; Изменение размера не прошло,
        mov     cx,msg1_length  ; сообщение об ошибке и на выход
        jmp     shell4

shell1: call    get_comspec     ; Получить спецификацию файла COMMAND.COM
        jnc     shell2          ; Переход, если он найден
```

Рис. 12.5. SHELL.ASM. Простой табличный интерпретатор команд, написанный на языке Microsoft Macro Assembler.

```
        mov     dx,offset msg3  ; В контексте не найдена переменная
        mov     cx,msg3_length  ; COMSPEC. Выдать сообщение об ошибке
        jmp     shell4          ; и на выход

shell2: mov     dx,offset shell3; Установить вектор Ctrl-C (Int 23h)
        mov     ax,cs           ; для обработчика этой программы
        mov     ds,ax           ; DS:DX = адрес обработчика
        mov     ax,2523h        ; Функция 25h = установить вектор
        int     21h             ; Переход в MS-DOS

        mov     ax,_DATA        ; Снова сделаем наш сегмент данных
        mov     ds,ax           ; адресуемым
        mov     es,ax

shell3:                         ; Главный цикл интерпретатора

        call    get_cmd         ; Получить команду пользователя

        call    intrinsic       ; Проверить, не внутренняя ли функция
        jnc     shell3          ; Да, уже выполнена

        call    extrinsic       ; Нет, передать COMMAND.COM,
        jmp     shell3          ; затем получить новую команду

shell4:                         ; Сюда,если обнаружена ошибка
                                ; DS:DX = адрес сообщения
                                ; CX = длина сообщения
        mov     bx,stderr       ; BX = дескриптор стандартной ошибки

        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS

        mov     ax,4c01h        ; Функция  4ch = завершение
                                ; с кодом возврата = 1
        int     21h             ; Переход в MS-DOS

shell   endp


intrisic proc  near             ; Поиск команды пользователя
                                ; в таблице "COMMANDS".
                                ; Если команда найдена, запустить
                                ; программу и вернуть флаг переноса = "ложь"
                                ; Если не найдена, флаг переноса = "истина"
                                ; При возврате флаг переноса = "истина"

        mov     si,offset_commands ; DS:SI = таблица команд

intr1:  cmp     byte ptr [si],0 ; Конец таблицы?
        je      intr7           ; Переход, найден конец таблицы
        mov     di,offset inp_buf  ; Нет, пусть DI = адрес
                                   ; команды пользователя

intr2:  cmp     byte ptr [di],blank; Убрать все лидирующие пробелы
        jne     intr3
```

Рис. 12.5.

```
        inc     di              ; Найден пробел, пропустить его
        jmp     intr2

intr3:  mov     al,[si]         ; Следующий символ из таблицы

        or      al,al           ; Конец строки?
        jz      intr4           ; Переход, вся строка совпала

        cmp     al,[di]         ; Сравнить с введенным символом
        jnz     intr6           ; Переход, найдено несоответствие

        inc     si              ; Продвинуть указатели строк
        inc     di
        jmp     intr3

intr4:  cmp     byte ptr [di],cr; Убедитесь, что команда пользователя
        je      intr5           ; имеет ту же длину...
        cmp     byte ptr [di],blanc ; Следующий символ во введенной команде
        jne     intr6           ; должен быть пробелом или возвратом каретки

intr5:  call    word ptr [si+1] ; Запустить COMMAND.COM

        clc                     ; Вернуть флаг переноса = "ложь",
        ret                     ; как флаг успешного завершения

intr6:  lodsb                   ; Ищем конец этой командной
        or      al,al           ; строки (нулевой байт)
        jnz     intr6           ; Еще не конец, цикл

        add     si,2            ; Перешагнем через адрес программы
        jmp     intr1           ; Проверим на соответствие следующую команду

intr7:  stc                     ; Соответствующей команды не нашлось, выход
        ret                     ; с флагом переноса = "истина"

intrisic endp

extrinsic proc near             ; Обработка внешней команды
                                ; путем передачи ее
                                ; программе COMMAND.COM
                                ; с хвостом команды " /C "

        mov     al,cr               ; Найти длину команды
        mov     cx,cmd_tail_length  ; путем поиска
        mov     di,offset cmd_tail+1; перевода каретки
        cld
        repnz scasb

        mov     ax,di               ; Вычислить длину хвоста команды
        sub     ax,offset cmd_tail+2; без перевода каретки
        mov     cmd_tail,al         ; и сохранить ее

                                    ; Установить адрес командного хвоста
        mov     word ptr par_cmd,offset cmd_tail
        call    exec                ; и запустить COMMAND.COM

extrinsic endp
```

Рис. 12.5.

```
get_cmd proc    near            ; Запрос пользователя, получить команду
                                ;
                                ; Выдать запрос оболочки
        mov     dx,offset prompt; DS:DX = адрес сообщения
        mov     cx,prompt_length; CX = длина сообщения
        mov     bx,stdout       ; BX = дескриптор  стандартного вывода
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS

                                ; Получить команду пользователя
        mov     dx,offset inp_buf ; DS:DX = входной буфер
        mov     cx,inp_buf_length ; CX = максимальная длина при чтении
        mov     bx,stdin        ; BX = дескриптор стандартного ввода
        mov     ah,3fh          ; Функция 40h = чтение
        int     21h             ; Переход в MS-DOS

        mov     si,offset inp_buf ; Свернуть вводимые символы нижнего
        mov     cx,inp_buf_length ; регистра в коды верхнего регистра
gcmd1:  cmp     byte ptr [si],'a' ; Проверить на 'a-z'
        jb      gcmd2           ; Переход, если не в этом диапазоне
        cmp     byte ptr [si],'z' ; Проверить на 'a-z'
        ja      gcmd2             ; Переход, если не в этом диапазоне
        sub     byte ptr [si],'a'-'A'; Преобразовать в коды верхнего регистра

gcmd2:  inc     si              ; Продвинуться по введенному
        loop    gcmd1
        ret                     ; Назад в вызывающую программу



get_cmd endp


get_comspec proc near           ; Получить местоположение COMMAND.COM
                                ; из контекстной переменной "COMSPEC="
                                ; Возвращает перенос = "ложь",
                                ; если найдена COMSPEC
                                ; Возвращает перенос = "истина",
                                ; если COMSPEC не найдена

        mov     si,offset com_var ; DS:SI = строка для поиска соответствия...
        call    get_env         ; Просмотр блока контекста
        jc      gcsp2           ; Переход, если COMSPEC не найдена

                                ; ES:DI указывает за "="
        mov     si,offset com_spec ; DS:SI = локальный  буфер

gcsp1:  mov     al,es:[di]      ; Скопировать переменную COMSPEC
        mov     [si],al         ; в локальный буфер
        inc     si
        inc     di
        or      al,al           ; Нулевой символ? (сбрасывает перенос)
        jnz     gcsp1           ; Нет, получить следующий символ

gcsp2:  ret                     ; Назад в вызывающую программу

get_comspec endp
```

Рис. 12.5.

```
get_env proc    near            ; Просмотреть контекст
                                ; Вызвать DS:SI = "NAME="
                                ; Использует содержимое "ENV_SEG"
                                ; Возвращает перенос = "ложь" и ES:DI,
                                ; указывающий на найденный параметр
                                ; Возвращает перенос = "истина", если
                                ; соответствия нет
        mov     es,env_seg      ; Получить сегмент контекста
        xor     di,di           ; Инициализировать относительный адрес
                                ; контекста
genv1:  mov     bx,si           ; Инициализировать указатель на имя
        cmp     byte ptr es:[di],0  ; Конец контекста?
        jne     genv2           ; Переход, конец не найден

        stc                     ; Соответствия нет, возврат
        ret                     ; с установленным переносом

genv2:  mov     al,[bx]         ; Получить символ из имени
        or      al,al           ; конец имени? (сбрасывает перенос)
        jz      genv3           ; Да, имя соответствует

        cmp     al,es:[di]      ; Сравнить с контекстом
        jne     genv4           ; Переход, если соответствия нет

        inc     bx              ; Продвинуть указатели контекста
        inc     di              ; и имени
        jmp     genv2

genv3:                          ; Соответствие найдено, перенос сброшен,
        ret                     ; ES:DI = переменная

genv4:  xor     al,al           ; Просмотр контекста вперед
        mov     cx,-1           ; в поисках нулевого байта
        cld
        repnz   scasb
        jmp     genv1           ; На сравнение следующей строки

get_env endp


exec    proc    near            ; Вызов функции EXEC MS DOS
                                ; для запуска COMMAND.COM

        mov     stkseg,ss       ; Сохранить указатель стека
        mov     stkptr,sp

                                ; Теперь выполнение COMMAND.COM
        mov     dx,offset com_spec  ; DS:DX = имя файла
        mov     bx,offset par_blk   ; ES:BX = блок параметров
        mov     ax,4b00h            ; Функция 4bh = EXEC
                                ; Подфункция 0 =
                                ; загрузить и выполнить
        int     21h             ; Переход в MS-DOS

        mov     ax,_DATA        ; Сделаем сегмент данных
        mov     ds,ax           ; снова адресуемым
        mov     es,ax
```

Рис. 12.5.

```
        cli                     ; (Из-за ошибок в некоторых 8088)
        mov     ss,stkseg       ; Восстановить указатель стека
        mov     sp,stkptr
        sti                     ; (Из-за ошибок в некоторых 8088)

        jnc     exec1           ; Переход, если ошибок нет

                                ; Вывести сообщение об ошибке
        mov     dx,offset msg2  ; DS:DX = адрес сообщения
        mov     cx,msg2_length  ; CX = длина сообщения
        mov     bx,stderr       ; BX = дескриптор стандартной ошибки
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS

exec1:  ret                     ; Назад в вызывающую программу

exec    endp

cls_cmd proc    near            ; Внутренняя команда CLS
        mov     dx,offset cls_str ; Послать Esc-последовательность
        mov     cx,cls_str_length ; ANSI для очистки
        mov     bx,stdout       ; экрана
        int     21h
        ret

cls_cmd endp


dos_cmd proc    near            ; Внутренняя команда DOS

                                ; Установить нулевой хвост команды
        mov     word ptr par_cmd,offset nultail
        call    exec            ; и запустить COMMAND.COM
        ret

dos_cmd endp

exit_cmd proc   near            ; Внутренняя команда EXIT

        mov     ax,4c00h        ; Вызвать функцию завершения
        int     21h             ; MS-DOS с нулевым кодом
                                ; возврата

exit_cmd endp

_TEXT   ends


STACK   segment para stack 'STACK'  ; Объявить сегмент стека

        dw      64 dup (?)

STACK   ends

_DATA   segment word public 'DATA'

commands equ $                  ; Таблица "внутренних" команд
```

Рис. 12.5.

```
                                ; Каждая запись - строка ASCII,
                                ; за которой следует относительный адрес
                                ; процедуры, которая должна выполниться,
                                ; как результат этой команды
        db      'CLS',0
        dw      cls_cmd

        db      'DOS',0
        dw      dos_cmd

        db      'EXIT',0
        dw      exit_cmd

        db      0               ; Конец таблицы

com_var db      'COMSPEC=',0    ; Переменная контекста

                                ; Спецификация файла COMMAND.COM
com_spec db     80 dup (0)      ; из контекста COMSPEC=

nultail db      0,cr            ; Нулевой хвост команды для
                                ; возбуждения COMMAND.COM,
                                ; как второй оболочки

cmd_tail db     0,'/C '         ; Хвост команды для возбуждения
                                ; COMMAND.COM, как транзитной программы
inp_buf db      80 dup (0)      ; Командная строка для стандартного ввода

inp_buf_length equ $-inp_buf
cmd_tail_length equ $-cmd_tail-1

prompt  db      cr,lf,'sh: '    ; Запрос пользователя из SHELL
prompt_length equ $-prompt

env_seg dw      0               ; Сегмент блока контекста

msg1    db      cr,lf
        db      'Не могу освободить память.'
        db      cr,lf
        msg1_length equ $-msg1

msg2    db      cr,lf
        db      'Ошибка вызова COMMAND.COM'
        db      cr,lf
msg2_length equ $-msg2

msg3    db      cr,lf
        db      'В контексте отсутствует переменная COMSPEC'
        db      cr,lf
msg3_length equ $-msg3

cls_str db      escape,'[2J'    ; Esc-последовательность ANSI
cls_str_length $-cls_str        ; для очистки экрана

                                ; Блок параметров EXEC
parblk  dw      0               ; Сегмент контекста
par_cmd dd      cmd_tail        ; Командная строка
        dd      fcb1            ; Блок управления файлом #1
```

Рис. 12.5.

```
        dd      fcb2            ; Блок управления файлом #2

fcb1    db      0               ; Блок управления файлом #1
        db      11 dup (' ')
        db      25 dup (0)

fcb2    db      0               ; Блок управления файлом #2
        db      11 dup (' ')
        db      25 dup (0)

stkseg  dw      0               ; Первоначальное содержимое SS
stkptr  dw      0               ; Первоначальное содержимое SP

_DATA   ends

        end     shell
```

Рис. 12.5.

Программа SHELL является таблично-ориентированной; она легко расширяется, предоставляя пользователю гибкий настраиваемый интерфейс, который можно с успехом использовать едва ли не для любого приложения. Получив управление, программа SHELL выводит на экран запрос sh: и ожидает от пользователя ввода входной строки. После того как пользователь набрал на клавиатуре строку, заканчивающуюся возвратом каретки, SHELL пытается сопоставить первый элемент строки с содержимым своей таблицы внутренних команд. Если соответствие обнаружено, SHELL вызывает соответствующую программу. Если соответствие не обнаружено, SHELL вызывает функцию MS-DOS EXEC и передает строку, введенную пользователем, интерпретатору COMMAND.COM (с ключом/C), используя COMMAND.COM в качестве процессора транзитных команд под своим управлением.

Как видно из листингов, SHELL «знает» в точности три внутренние команды:

Команда	**Действие**
CLS	Использует стандартную управляющую последовательность ANSI для очистки экрана и установки курсора в исходное положение
DOS	Запускает копию COMMAND.COM
EXIT	Завершает работу SHELL, возвращая управление системой следующему (нижнему) интерпретатору команд

Вы можете без труда добавить новые внутренние команды в любой из приведенных вариантов программы SHELL. Просто напишите программу процедуры с требуемыми функциями и вставьте имя этой процедуры вместе с текстовой строкой, определяющей команду, в таблицу COMMANDS. Вдобавок вы можете легко запретить SHELL передавать COMMAND.COM определенные «опасные» команды (вроде MKDIR или ERASE), попросту включив их в таблицу внутренних команд вместе с адресами подпрограмм вывода аварийных сообщений.

Ход выполнения обоих вариантов программы SHELL выглядит следующим образом:

1. Программа вызывает прерывание MS-DOS Int 21H с функцией 4AH (Изменить размер блока памяти), чтобы сократить потребление памяти и предоставить максимум пространства памяти интерпретатору COMMAND.COM, если он будет выполняться в виде оверлея. (Это делается в явной форме только в варианте на языке ассемблера. Ради упрощения примера количество резервируемых параграфов записано в виде

«щедрой» константы, в то время как в действительности эта величина должна определяться в процессе выполнения исходя из размера и расположения различных сегментных программ.)

2. Программа просматривает окружение в поисках переменной COMSPEC, определяющей местоположение выполнимой копии COMMAND.COM. Если переменная COMSPEC не найдена, программа выводит аварийное сообщение и завершается.

3. Программа загружает адрес собственного обработчика прерываний в вектор Ctrl-C (Int 23H), чтобы не потерять контроль над системой, если пользователь введет Ctrl-C или Ctrl-Break.

4. Программа направляет текст запроса на стандартное устройство вывода.

5. Программа считывает со стандартного устройства ввода буферизованную строку с командой пользователя.

6. Программа ищет соответствие первого элемента входной строки (отделенного пробелами) внутренним командам своей таблицы. Если соответствие обнаружено, выполняется соответствующая процедура.

7. Если в таблице внутренних команд искомый элемент не обнаружен, программа составляет хвост командной строки, подсоединяя к ключу /C введенную пользователем команду, и вызывает затем с помощью функции EXEC копию COMMAND.COM, передавая адрес составленного хвоста команды через блок параметров EXEC.

8. Программа повторяет шаги с 4-го по 7-й до тех пор, пока пользователь не введет команду EXIT, относящуюся к числу внутренних команд и заставляющую SHELL завершить выполнение.

В существующем виде SHELL позволяет интерпретатору COMMAND.COM получить полную копию текущего окружения. В некоторых приложениях, однако, может оказаться более удобным (или более безопасным) передавать модифицированную копию блока окружения, чтобы вторичная копия COMMAND.COM не имела доступа к определенной информации.

Использование EXEC для загрузки оверлеев

Загрузка с помощью функции EXEC оверлеев оказывается гораздо более простой операцией, чем использование EXEC для запуска другой программы. Оверлей можно оформить либо в виде образа памяти (.COM), либо в виде перемещаемого файла (.EXE), причем типы оверлея и загружающей его программы могут различаться. Для загрузки и запуска оверлея основная программа, называемая корневым сегментом, должна выполнить следующие шаги:

1. Обеспечить наличие свободного блока памяти для получения оверлея. Программа, вызывающая функцию EXEC, должна иметь в своем распоряжении блок памяти для оверлея.

2. Заполнить блок параметров оверлея для передачи функции EXEC. Этот блок содержит сегментный адрес блока памяти, предназначенного для получения оверлея, а также сегментную константу перемещения для содержимого оверлейного файла (если это файл .EXE). Обычно эти величины совпадают.

3. Вызвать функцию MS-DOS EXEC для загрузки оверлея, для чего выполнить Int 21H, подготовив регистры следующим образом:

AH = 4BH
AL = 03H (подфункция EXEC для загрузки оверлеев)
DS:DX = сегмент:относительный_адрес пути к оверлейному файлу
ES:BX = сегмент:относительный_адрес блока параметров оверлея

При возврате из функции EXEC, если оверлей найден и загружен, флаг переноса сбрасывается. Если файл не найден или встретилась другая ошибка, флаг переноса устанавливается.

4. Выполнить программные строки оверлея, передав на них управление с помощью дальнего вызова. Оверлей следует составлять таким образом, чтобы в начале загруженного модуля располагались либо точка входа, либо указатель на точку входа. Этот прием позволит вам работать над корневым модулем и модулем оверлея независимо, так как корневой модуль в этом случае не обладает «магическими» сведениями об адресах внутри сегмента оверлея.

Для того чтобы пользователь случайно не запустил оверлей непосредственно с командной строки, файлу оверлея следует назначить расширение, отличающееся от .COM и .EXE. Удобнее всего соотнести оверлеи с их корневым сегментом, назначив им то же имя файла, но другое расширение, например .OVL или .OV1, .OV2 и т.д.

На рис. 12.6 показано использование функции EXEC для запуска и выполнения оверлея.

```
.
.
.
                        ; Выделить память для оверлея
mov     bx,100h         ; Получить 64 кбайт (4096 параграфов)
mov     ah,48h          ; Функция 48h = выделить блок
int     21h             ; Переход в MS-DOS
jc      error           ; Переход, если выделение не прошло

mov     pars,ax         ; Установить адрес загрузки для оверлея
mov     pars+2,ax       ; Установить сегмент перемещения для оверлея

                        ; Установить сегмент точки входа
mov     word ptr entry+2,ax

mov     stkseg,ss       ; Сохранить указатель стека корневого сегмента
mov     stkptr,sp

mov     ax,ds           ; Установить ES = DS
mov     es,ax

mov     dx,offset oname ; DS:DX = путь к файлу с оверлеем
mov     bx,offset pars  ; ES:BX = блок параметров
mov     ax,4b03h        ; Функция 4bh, подфункция 03h
int     21h             ; Переход в MS-DOS

mov     ax,_DATA        ; Снова сделаем наш сегмент данных
mov     ds,ax           ; адресуемым
mov     es,ax

cli                     ; (Из-за ошибки в некоторых ранних МП 8088)
mov     ss,stkseg       ; Восстановить указатель стека
mov     sp,stkptr
sti                     ; (Из-за ошибки в некоторых ранних МП 8088)

jc      error           ; Переход, если функция EXEC не выполнилась

                        ; В противном случае EXEC выполнилась успешно...
push    ds              ; Сохраним наш сегмент данных
```

Рис. 12.6. Структура программы, осуществляющей загрузку и выполнение оверлея с помощью функции EXEC. Файл оверлея может быть как в формате .COM, так и в формате .EXE.

```
call    dword ptr entry ; Теперь вызовем оверлей
pop     ds              ; Восстановим наш сегмент данных
.
.
.

oname   db      'OVERLAY.OVL',0 ; Путь к файлу с оверлеем

pars    dw      0               ; Адрес загрузки файла (сегментный)
        dw      0               ; Константа перемещения (сегментная) для файла

entry   dd      0               ; Входная точка для оверлея

stkseg  dw      0               ; Сохранить регистр SS
stkptr  dw      0               ; Сохранить регистр SP
```

Рис. 12.6.

ГЛАВА 13. ОБРАБОТЧИКИ ПРЕРЫВАНИЙ

Прерывания – это сигналы, заставляющие центральный процессор (ЦП) компьютера приостановить выполнение текущей работы и перейти на выполнение программы, называемой обработчиком прерываний (interrupt handler). Переход этот осуществляется за малое время с помощью специально разработанных аппаратных средств. Обработчик прерываний определяет причину прерывания, выполняет запланированные действия, после чего возвращает управление приостановленному процессу.

Обычно прерывания вызываются событиями, внешними по отношению к ЦП и требующими немедленных действий. К таким событиям, в частности, относятся:

- Завершение операции ввода-вывода
- Обнаружение аппаратного сбоя
- «Катастрофы» (например, отказ питания)

Для обеспечения эффективного обслуживания прерываний большинство современных процессоров поддерживает механизм типов или уровней прерываний. Каждому типу обычно соответствует ячейка в памяти, называемая вектором прерывания, которая определяет местоположение программы обработчика прерываний данного типа. Такая система ускоряет обработку прерывания, так как компьютер получает возможность непосредственной передачи управления соответствующей программе; отпадает необходимость в центральной программе, которая определяет источник прерывания, затрачивая на это драгоценные машинные циклы. Концепция типов прерываний позволяет к тому же назначать прерываниям приоритеты, чтобы в случае одновременного возникновения нескольких прерываний наиболее важное могло быть обработано в первую очередь.

Процессоры компьютеров, поддерживающие систему прерываний, должны иметь средства блокирования прерываний на время выполнения критических участков программы. Иногда ЦП блокирует уровни прерываний избирательно, но чаще блокирование имеет глобальный характер. Пока идет обслуживание прерывания, ЦП маскирует все остальные прерывания того же и более низких уровней до тех пор, пока программа активного обработчика не завершится; точно так же ЦП может приостановить выполнение программы обработчика, если другое прерывание с более высоким приоритетом потребовало обслуживания. Некоторые процессоры могут даже отличать избирательное маскирование прерываний (прерывания фиксируются, но их обработка откладывается) от запрета прерываний (прерывания просто отбрасываются).

Создание обработчиков прерываний традиционно рассматривается как одна из наиболее таинственных программистских задач, посильная лишь избранным кругам системных программистов. В действительности написание программы обработчика прерываний – само по себе дело довольно простое. Хотя детали процедуры должны, конечно, учитывать характеристики конкретного процессора и операционной системы, общие указания, приведенные ниже, применимы почти к любой компьютерной системе.

Программа, приступающая к обработке прерываний, должна выполнить следующие действия:

1. Запретить прерывания, если они были ранее разрешены, чтобы предотвратить их возникновение во время процедуры модификации векторов прерываний.
2. Инициализировать вектор конкретного источника прерываний, чтобы он указывал на обработчик прерываний, относящийся к данной программе.
3. Убедиться в том, что все остальные векторы разрешенных ранее прерываний указывают на обработчики прерываний, имеющие смысл для данной задачи.
4. Снова разрешить прерывания.

Сам обработчик прерываний должен выполнять простую, но жестко определенную последовательность действий:

1. Сохранить системный контекст (регистры, флаги и все остальное, что может быть модифицировано обработчиком и не сохраняется автоматически центральным процессором).
2. Заблокировать все прерывания, возникновение которых в процессе выполнения программы обработчика может привести к конфликту. (Часто это делается автоматически аппаратурой компьютера.)
3. Разрешить все прерывания, которые имеют право возникать во время обработки данного прерывания.
4. Определить источник прерывания.
5. Выполнить запланированную обработку прерывания: получить и сохранить данные, полученные через последовательный порт, установить флаг, указывающий на завершение передачи сектора диска, и т. д.
6. Восстановить системный контекст.
7. Разрешить все прерывания, которые были запрещены на время обработки данного прерывания.
8. Возобновить выполнение прерванного процесса.

Как и в случае любой другой программы, ключ к успешному написанию обработчика прерываний лежит в тщательном программировании, учитывающем все возможные источники неприятностей. Основная причина, по которой обработчики прерываний заслужили такую мистическую репутацию, заключается в сложности их отладки при наличии скрытых ошибок. Из-за того что прерывания возникают асинхронно, т. е. инициируются внешними событиями безотносительно к состоянию выполняемого процесса, ошибки в обработчиках прерываний приводят к совершенно непредсказуемому поведению всей системы.

Прерывания и семейство Intel 80 × 86

Семейство микропроцессоров Intel 80 × 86 поддерживает 256 уровней приоритетных прерываний, вызываемых событиями трех типов:

- Внутренние аппаратные прерывания
- Внешние аппаратные прерывания
- Программные прерывания

Внутренние аппаратные прерывания

Внутренние аппаратные прерывания, иногда называемые отказами (faults), генерируются определенными событиями, возникающими в процессе выполнения программы, например попыткой деления на нуль. Закрепление за такими событиями определенных номеров прерываний зашито в процессоре и не может быть изменено (рис. 13.1).

Внешние аппаратные прерывания

Внешние аппаратные прерывания инициируются контроллерами периферийного оборудования или сопроцессорами (например, 8087/80287). Источники сигналов прерываний подключаются либо к выводу немаскируемых прерываний процессора (NMI), либо к выводу маскируемых прерываний (INTR). Линия NMI обычно предназначается для прерываний, вызываемых катастрофическими событиями, такими, как ошибка четности памяти или авария питания.

Вместо непосредственного подключения к ЦП прерывания от внешних устройств могут поступать в процессор через специальное устройство – программируемый контроллер прерываний (PIC) 8259A. ЦП управляет контроллером через набор портов ввода-вывода, а контроллер в свою очередь сигнализирует процессору через вывод INTR. PIC предоставляет возможность программно разрешать и запрещать прерывания от конкретных устройств, а также назначать им приоритеты.

Один контроллер прерываний обслуживает только 8 уровней прерываний. Однако контроллеры можно соединять последовательно с образованием древовидной структуры, которая может обеспечить обслуживание любого требуемого числа уровней

Уровень прерывания	Адрес вектора	Источник прерывания	8086/88	80286	80386
00H	00H–03H	Деление на нуль	×	×	×
01H	04H–07H	Одиночный шаг	×	×	×
02H	08H–0BH	Немаскируемое прерывание (NMI)	×	×	×
03H	0CH–0FH	Точка останова	×	×	×
04H	10H–13H	Переполнение	×	×	×
05H	14H–17H	Превышение границы BOUND		×	×
06H	18H–1BH	Неправильный код операции		×	×
07H	1CH–1FH	Расширение процессора отсутствует		×	×
08H	20H–23H	Двойной отказ		×	×
09H	24H–27H	Перекрытие сегмента		×	×
0AH	28H–2BH	Неправильный сегмент состояния задачи		×	×
0BH	2CH–2FH	Сегмент отсутствует		×	
0CH	30H–33H	Перекрытие сегмента стека		×	×
0DH	34H–37H	Отказ общей защиты		×	×
0EH	38H–3BH	Отказ страницы			×
0FH	3CH–3FH	Зарезервирован			
10H	40H–43H	Ошибка арифметического сопроцессора		×	×
11H–1FH	44H–7FH	Зарезервированы		×	×

Рис. 13.1. Внутренние прерывания (отказы) в микропроцессорах Intel 8086/88, 80286 и 80386.

прерываний. Например, машины на базе микропроцессоров 80286 и 80386 с PC/AT-совместимой архитектурой используют два связанных вместе контроллера прерываний, что дает возможность независимой работы с 16 уровнями прерываний.

Прерывания INTR можно глобально разрешить или запретить с помощью команд процессора STI и CLI. Как и следует ожидать, эти команды не оказывают влияния на прерывания, поступающие через вывод NMI.

Изготовители компьютерных систем так же, как и изготовители периферийного оборудования, назначают внешним устройствам определенные уровни контроллера прерываний 8259A. Эти назначения выполняются в виде физических электрических соединений и не могут быть изменены программным образом.

Программные прерывания

Любая программа может инициировать синхронное программное прерывание просто путем выполнения команды INT. MS-DOS использует для взаимодействия со своими модулями и прикладными программами прерывания от 20H до 3FH. (Например, доступ к диспетчеру функций MS-DOS осуществляется выполнением Int 21H.) Программы BIOS, хранящиеся в ПЗУ, и прикладные программы IBM PC используют другие прерывания, с большими или меньшими номерами (рис. 13.2). Это распределение номеров прерываний условно и никаким образом не закреплено аппаратно.

Прерывание	Использование	Машины
00H	Деление на нуль	PC, AT, PS/2
01H	Одиночный шаг	PC, AT, PS/2
02H	NMI	PC, AT, PS/2
03H	Точка останова	PC, AT, PS/2
04H	Переполнение	PC, AT, PS/2
05H	ПЗУ BIOS, PrintScreen	PC, AT, PS/2
	Превышение BOUND	AT, PS/2
06H	Зарезервировано	PC
	Неправильный код операции	AT, PS/2
07H	Зарезервировано	PC
	80287/80387 отсутствует	AT, PS/2
08H	IRQ0, такт таймера	PC, AT, PS/2
	Двойной отказ	AT, PS/2
09H	IRQ1, клавиатура	PC, AT, PS/2
	80287/80387, перекрытие сегмента	AT, PS/2
0AH	IRQ2, зарезервировано	PC
	IRQ2, каскадированное от ведомого 8259A	AT, PS/2
	Неправильный сегмент состояния задачи (TSS)	AT, PS/2
0BH	IRQ3, последовательная связь (COM2)	PC, AT, PS/2
	Сегмент отсутствует	AT, PS/2
0CH	IRQ4, последовательная связь (COM1)	PC, AT, PS/2
	Переполнение сегмента стека	AT, PS/2
0DH	IRQ5, жесткий диск	PC
	IRQ5, параллельный принтер (LPT2)	AT
	Зарезервировано	PS/2
	Отказ общей защиты	AT, PS/2
0EH	IRQ6, гибкий диск	PC, AT, PS/2
	Отказ страницы	AT, PS/2
0FH	IRQ7, параллельный принтер	PC, AT, PS/2
10H	ПЗУ BIOS, драйвер дисплея	PC, AT, PS/2

Рис. 13.2. Прерывания с закрепленными функциями на машинах IBM PC, PC/AT и PS/2 и их аналогах.

Заметьте, что ПЗУ BIOS фирмы IBM использует несколько прерываний в диапазоне 00H – 1FH, несмотря на то, что они были зарезервированы фирмой Intel для отказов центрального процессора. Номера IRQ соответствуют уровням приоритета контроллера прерывания Intel 8259A.

	Отказ арифметического сопроцессора	AT, PS/2
11H	ПЗУ BIOS, проверка оборудования	PC, AT, PS/2
12H	ПЗУ BIOS, размер обычной памяти	PC, AT, PS/2
13H	ПЗУ BIOS, драйвер диска	PC, AT, PS/2
14H	ПЗУ BIOS, коммуникационный драйвер	PC, AT, PS/2
15H	ПЗУ BIOS, драйвер кассеты	PC
	ПЗУ BIOS, расширения системы ввода-вывода	AT, PS/2
16H	ПЗУ BIOS, драйвер клавиатуры	PC, AT, PS/2
17H	ПЗУ BIOS, драйвер принтера	PC, AT, PS/2
18H	ПЗУ Бейсик	PC, AT, PS/2
19H	ПЗУ BIOS, начальный загрузчик	PC, AT, PS/2
1AH	ПЗУ BIOS, время дня	AT, PS/2
1BH	ПЗУ BIOS, Ctrl-Break	PC, AT, PS/2
1CH	ПЗУ BIOS, такт таймера	PC, AT, PS/2
1DH	ПЗУ BIOS, таблица параметров дисплея	PC, AT, PS/2
1EH	ПЗУ BIOS, параметры гибкого диска	PC, AT, PS/2
1FH	ПЗУ BIOS, шрифт (символы 80H-FFH)	PC, AT, PS/2
20H	MS-DOS, процесс завершения	
21H	MS-DOS, диспетчер функций	
22H	MS-DOS, адрес завершения	
23H	MS-DOS, адрес обработчика Ctrl-C	
24H	MS-DOS, адрес обработчика критической ошибки	
25H	MS-DOS, абсолютное чтение диска	
26H	MS-DOS, абсолютная запись на диск	
27H	MS-DOS, завершение и сохранение в памяти	
28H	MS-DOS, пустое прерывание	
29H	MS-DOS, зарезервировано	
2AH	MS-DOS, сетевое перенаправление	
2BH–2EH	MS-DOS, зарезервировано	
2FH	MS-DOS, мультиплексорное прерывание	
30H–3FH	MS-DOS, зарезервировано	
40H	ПЗУ BIOS, драйвер гибкого диска (если установлен жесткий диск)	PC, AT, PS/2
41H	ПЗУ BIOS, параметры жесткого диска	PC
	ПЗУ BIOS, параметры жесткого диска (дисковод 0)	AT, PS/2
42H	ПЗУ BIOS, подразумеваемый драйвер дисплея (если установлен EGA)	PC, AT, PS/2
43H	EGA, MCGA, VGA, таблица символов	PC, AT, PS/2
44H	ПЗУ BIOS, шрифт (символы 00H–7FH)	PCjr
46H	ПЗУ BIOS, параметры жесткого диска (дисковод 1)	AT, PS/2
4AH	ПЗУ BIOS, обработчик будильника	AT, PS/2
5AH	Адаптер кластера	PC, AT
5BH	Используется программой кластера	PC, AT
60H–66H	Прерывания пользователя	PC, AT, PS/2
67H	LIM EMS, драйвер	PC, AT, PS/2
68H–6FH	Не назначено	
70H	IRQ8, КМОП-часы реального времени	AT, PS/2
71H	IRQ9, программная передача на IRQ2	AT, PS/2
72H	IRQ10, зарезервировано	AT, PS/2
73H	IRQ11, зарезервировано	AT, PS/2
74H	IRQ12, зарезервировано	AT
	IRQ12, мышь	PS/2
75H	IRQ13, арифметический сопроцессор	AT, PS/2
76N	IRQ14, контроллер жесткого диска	AT, PS/2
77H	IRQ15, зарезервировано	AT, PS/2
78H–7FH	Не назначено	
80H–FOH	Бейсик	PC, AT, PS/2
F1H–FFH	Не используется	PC, AT, PS/2

Рис. 13.2.

Таблица векторов прерываний

Нижние (с меньшими адресами) 1024 байт системной памяти носят название таблицы векторов прерываний. Каждый 4-байт элемент таблицы соответствует типу прерывания (от 0 до 0FFH) и содержит сегмент и относительный адрес обработчика прерывания данного уровня. Прерывания от 0 до 1FH (самые нижние уровни) используются как прерывания внутренней аппаратуры; MS-DOS использует прерывания от 20H до 3FH, все остальные прерывания могут использоваться внешней аппаратурой, системными драйверами и прикладными программами.

Если контроллер прерываний 8259A или другое устройство посылает в ЦП прерывание через вывод INTR, оно должно также поместить тип прерывания в виде 8-разрядного числа (от 0 до 0FFH) на системную магистраль для передачи в ЦП. Процессор, умножив это число на 4, определяет адрес используемого вектора прерывания.

```
pic_ctl equ     20h             ; Порт управления контроллером
                                ; прерываний 8259A
        .
        .
        .
        sti                     ; Снова разрешить прерывания,
        push    ax              ; сохранить регистры
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    bp
        push    ds
        push    es

        mov     ax,cs           ; Сделать локальные данные адресуемыми
        mov     ds,ax
        .                       ; Сюда нужно включить что-нибудь
        .                       ; соответствующее этому прерыванию
        .
        mov     al,20h          ; Пошлем EOI в контроллер прерываний 8259A
        mov     dx,pic_ctl
        out     dx,al

        pop     es              ; Восстановить регистры
        pop     ds
        pop     bp
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        iret                    ; Возобновить предыдущие операции
```

Рис. 13.3. Типичный обработчик аппаратных прерываний для семейства микропроцессоров 80 × 86.

В действительности обработчик прерываний должен сохранять и восстанавливать только те регистры, которые он будет модифицировать. Кроме того, если обработчик активно использует машинный стек, он должен сохранить и восстановить регистры SS и SP прерванного процесса и использовать собственный локальный стек.

Обслуживание прерываний

ЦП, обнаружив сигнал прерывания, помещает в машинный стек слово состояния программы (определяющее различные флаги ЦП), регистр программного сегмента (CS) и указатель команд (IP) и блокирует систему прерываний. Затем ЦП с помощью 8-разрядного числа, установленного на системной магистрали прерывающим устройством, извлекает из таблицы векторов адрес обработчика и возобновляет выполнение с этого адреса.

Обычно обработчик немедленно разблокирует систему прерываний (чтобы разрешить прохождение прерываний с более высокими приоритетами), сохраняет регистры, которые будут им использоваться, и обрабатывает прерывание за минимально возможное время. Некоторые внешние устройства требуют при этом специального сигнала подтверждения, свидетельствующего о том, что прерывание было опознано процессором.

Если прерывание поступило в ЦП через контроллер прерываний 8259A, обработчик прерываний должен послать специальный код, называемый концом прерывания (end of interrupt, EOI). Этот код поступает в контроллер прерываний через его порт управления и оповещает контроллер о завершении обработки прерывания (EOI не оказывает никакого действия на сам ЦП). Наконец, обработчик прерываний выполняет специальную команду IRET (Interrupt Return, Возврат из прерывания), которая восстанавливает первоначальное состояние флагов ЦП, регистра CS и указателя команд (рис. 13.3).

Состояние системы в момент передачи управления обработчику прерываний совершенно не зависит от того, было ли прерывание возбуждено внешним устройством или явилось результатом выполнения программой команды INT. Это обстоятельство удобно использовать при написании и тестировании обработчиков внешних прерываний, отладку которых можно почти полностью выполнить, возбуждая их простыми программными средствами.

ОБРАБОТЧИКИ ПРЕРЫВАНИЙ И MS-DOS

Введение в вашу программу обработчика прерываний приводит к существенной аппаратной зависимости. Незачем говорить (хотя я все же скажу об этом еще раз), что в прикладных программах, работающих в среде MS-DOS, следует по возможности избегать аппаратной зависимости, чтобы обеспечить переносимость программ на любую машину с той или иной из нынешних версий MS-DOS, а также совместимость с будущими версиями операционной системы.

Имеются, однако, серьезные основания для разработки собственного обработчика прерываний, предназначенного для использования в среде MS-DOS:

- Чтобы заменить стандартный обработчик MS-DOS для внутренних аппаратных прерываний (таких, как деление на нуль, превышение границы BOUND и т. д.).
- Чтобы заменить стандартный обработчик MS-DOS для определенных системных исключительных ситуаций, например обработчик критических ошибок или обработчик Ctrl-C.
- Чтобы связать ваш собственный обработчик прерываний со стандартным системным обработчиком для внешнего устройства и выполнять в ответ на прерывание как системные, так и ваши собственные действия (типичным примером таких прерываний являются прерывания от системного таймера).
- Чтобы обслуживать прерывания, не поддерживаемые стандартными драйверами устройств MS-DOS (таких, как последовательный порт, который в режиме прерываний можно использовать на гораздо больших скоростях, чем в режиме опроса).

● Чтобы обеспечить связь программы, которая остается резидентной после завершения, с другими прикладными программами.

Для установки корректных обработчиков прерываний таким образом, чтобы они не вступали в конфликт с функциями операционной системы или другими обработчиками прерываний, MS-DOS предоставляет специальные средства:

Функция	**Действие**
Int 21H, функция 25H	Установить вектор прерывания
Int 21H, функция 35H	Получить вектор прерывания
Int 21H, функция 31H	Завершить и оставить резидентной

Эти функции дают возможность анализировать или модифицировать содержимое таблицы системных векторов прерываний и резервировать память для использования обработчиком, не вступая в конфликты с другими процессами в системе и не нарушая правила использования памяти. В разд. 2 этой книги «Справочник по функциям MS-DOS» приведено подробное описание этих функций с программными примерами.

В системе MS-DOS на функционирование обработчиков внешних аппаратных прерываний накладывается ряд весьма жестких ограничений:

● В силу отсутствия свойства повторной входимости у версий MS-DOS обработчик аппаратных прерываний в процессе обработки прерывания не должен вызывать функции MS-DOS.

● Как только обработчик получает управление, он должен немедленно разблокировать прерывания, чтобы не нарушать работу других устройств и не снижать точность системных часов.

● Программа должна обращаться к контроллеру прерываний с большой осторожностью. Обращение к нему допустимо только в том случае, когда программа является единственным процессом в системе, связанным с конкретным уровнем прерываний. Кроме того, важно, чтобы обработчик перед выполнением команды IRET послал в контроллер прерываний код конца прерывания; в противном случае обработка следующих прерываний для того же и более низких уровней прерываний будет заблокирована.

Ограничения, накладываемые на функционирование обработчиков прерываний, заменяющих стандартные обработчики MS-DOS для внутренних аппаратных прерываний или системных исключительных состояний (таких, как Ctrl-C или критическая ошибка), не столь жестки, однако программировать обработчики следует все же с большой осторожностью, чтобы не допустить разрушения системных таблиц или перевода операционной системы в неустойчивое состояние.

При составлении программы драйвера прерываний следует иметь в виду следующие правила:

● Для модификации вектора прерывания используйте прерывание Int 21H с функцией 25H (Установить вектор прерывания); не записывайте прямо в таблицу векторов прерываний.

● Если ваша программа не единственный процесс в системе, использующий данный уровень прерываний, то после выполнения собственной обработки прерывания вернитесь в предыдущий обработчик, к которому привязан ваш.

● Если ваша программа не остается резидентной в памяти, получите и сохраните текущее состояние вектора прерывания перед его модификацией, а при завершении программы восстановите исходное содержимое.

● Если ваша программа остается резидентной в памяти, используйте одну из функций завершения и сохранения в памяти (предпочтительно Int 21H с функцией 31H), чтобы зарезервировать для вашего обработчика требуемый объем памяти.

● Если вы собираетесь обрабатывать аппаратные прерывания, постарайтесь сократить время, в течение которого прерывания заблокированы, а также длину программы обработки до абсолютного минимума. Не забывайте, что даже после разблокировки прерываний командой STI прерывания того же и более низких приоритетов остаются заблокированными, если в системе используется контроллер прерываний 8295A.

ZERODIV, ПРИМЕР ОБРАБОТЧИКА ПРЕРЫВАНИЙ

Листинг ZERODIV.ASM (рис. 13.4) иллюстрирует использование некоторых принципов и рекомендаций, упомянутых ранее. ZERODIV представляет собой обработчик прерываний для внутренних прерываний по делению на нуль (тип 0). ZERODIV загружается как файл .COM (обычно с помощью команды в системном файле AUTOEXEC), но затем остается постоянно резидентным в памяти все время, пока работает система.

Программа ZERODIV включает два основных участка: участок инициализации и обработчик прерываний.

```
        name    zdivide
        page    55,132
        title   ZERODIV--Divide-by-zero handler
                        ;Обработчик деления на ноль

;
; ZERODIV.ASM--Резидентный после загрузки обработчик
;              прерываний по делению на ноль
;
; Авторские права 1988 Рей Данкан
;
; Компоновка:     C>MASM ZERODIV;
;                 C>LINK ZERODIV;
;                 C>EXE2BIN ZERODIV.EXE ZERODIV.COM
;                 C>DEL ZERODIV.EXE
;
; Использование: C>ZERODIV
;

cr      equ     0dh             ; Код ASCII возврата каретки
lf      equ     0ah             ; Код ASCII перевода строки
beep    equ     07h             ; Код ASCII гудка
backsp  equ     08h             ; Код ASCII забоя

-TEXT   segment word public 'CODE'

        org     100H

        assume  cs:_TEXT,ds:_TEXT,es:_TEXT,ss:_TEXT

init    proc    near            ; Точка входа  во время загрузки

                                ; Захватить вектор для
                                ; прерывания ноль ...
```

Рис. 13.4. Простой пример обработчика прерываний для использования в среде MS-DOS.

ZERODIV остается постоянно резидентным в памяти и обслуживает внутренние прерывания ЦП при делении на нуль.

```
        mov     dx,offset zdiv  ; DS:DX = адрес дескриптора
        mov     ax,2500h        ; Функция 25h = установить вектор
                                ; Тип прерывания = 0
        int     21h             ; Переход в MS-DOS

                                ; Напечатать сообщение-заголовок
        mov     dx,offset msg1  ; DS:DX = адрес сообщения
        mov     ah,9            ; Функция 09h = вывод строки
        int     21h             ; Переход в MS-DOS

                                ; DX = число резервируемых параграфов
        mov     dx,((offset pgm_len+15)/16)+10h
        mov     ax,3100h        ; Функция 31h = завершить и
                                ; оставить в памяти
        int     21h             ; Переход в MS-DOS

init    endp

zdiv    proc    far             ; Это обработчик прерывания
                                ; по делению на ноль

        sti                     ; Разрешить прерывания

        push    ax              ; Сохранить регистры
        push    bx
        push    cx
        push    dx
        push    si
        push    di
        push    bp
        push    ds
        push    es

        mov     ax,cx           ; Сделаем данные адресуемыми
        mov     ds,ax

                                ; Выдать сообщение
                                ; "Продолжить или Завершить?"
        mov     dx,offset msg2  ; DS:DX = адрес сообщения
        mov     ah,9            ; Функция 09 = вывод строки
        int     21h             ; Переход в MS-DOS

zdiv1:  mov     ah,1            ; Функция 01h = чтение с клавиатуры
        int     21h             ; Переход в MS-DOS

        or      al,20h          ; Перевести символ на нижний регистр

        cmp     al,'c'          ; Это C или Q?
        je      zdiv3           ; Переход, это C

        cmp     al,'q'
        je      zdiv2           ; Переход, это Q

                                ; Недопустимый ввод, послать гудок
                                ; и стереть символ
        mov     dx,offset msg3  ; DS:DX = адрес сообщения
        mov     ah,9            ; Функция 09h = вывод строки
```

Рис. 13.4.

```
        int     21h             ; Переход в MS-DOS

        jmp     zdiv1           ; Попробуем снова

zdiv2:                          ; Пользователь выбрал "Завершить"
        mov     ax,4cffh        ; Закончить текущую программу
        int     21h             ; с кодом возврата = 255

zdiv3:                          ; Пользователь выбрал "Продолжить"
                                ; Послать пару CR-LF
        mov     dx,offset msg4  ; DS:DX = адрес сообщения
        mov     ah,9            ; Функция 09h = вывод строки
        int     21h             ; Переход в MS-DOS

                                ; Это какой тип ЦП?
        xor     ax,ax           ; Чтобы узнать это, мы отправим
        push    ax              ; ноль в регистр флагов ЦП и
        popf                    ; посмотрим, что получится
        pushf
        pop     ax
        and     ax,0f000h       ; МП 8086/8088 устанавливает
        cmp     ax,0f000h       ; биты 12 - 15
        je      zdiv5           ; Переход, если 8086/8088

                                ; В противном случае мы должны подправить
                                ; адрес возврата, чтобы обойти
                                ; команду деления...
        mov     bp,sp           ; Сделаем стек адресуемым

        lds     bx,[bp+18]      ; Получить адрес команды,
                                ; дающей ошибку

        mov     bl,[bx+1]       ; Получить адресуемый байт
        and     bx,0c7h         ; Изолировать поля mod & r/m

        cmp     bl,6            ; mod 0, r/m 6 = прямая адресация
        jne     zdiv4           ; Непрямая, переход

        add     word ptr [bp+18],4
        jmp     zdiv5

zdiv4:  mov     cl,6            ; В противном случае изолировать
        shr     bx,cl           ; поле mod и получить из таблицы
        mov     bl,cs:[bx+itab] ; размер команды
        add     [bp+18],bx

zdiv5:  pop     es              ; Восстановить регистры
        pop     ds
        pop     bp
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        iret                    ; Возврат из прерывания

zdiv    endp
```

Рис. 13.4.

```
msg1    db      cr,lf           ; Сообщение-заголовок времени загрузки
        db      'Прерывание по делению на ноль'
        db      'Обработчик установлен. '
        db      cr,lf,'$'

msg2    db      cr,lf,lf        ; Сообщение времени прерывания
        db      'Обнаружено деление на ноль: '
        db      cr,lf,'Продолжить или закончить (C/Q)? '
        db      '$'

msg3    db      beep            ; Используется при недопустимом вводе
        db      backsp,' ',backsp,'$'

msg4    db      cr,lf,'$'       ; Возврат каретки - перевод строки

                                ; Таблица размеров команд
itab    db      2               ; mod = 0
        db      3               ; mod = 1
        db      4               ; mod = 2
        db      2               ; mod = 3

pgm_len equ     $-init          ; Длина программы

_TEXT   ends

        end     init
```

Рис. 13.4.

Процедура инициализации (в листинге программы она обозначена init) выполняется только один раз, когда программа ZERODIV запускается с уровня MS-DOS. Процедура init захватывает вектор прерывания типа 0, выводит начальное сообщение – идентификатор программы, выполняет функцию завершения и сохранения в памяти и завершается выходом в MS-DOS. Такой выход в MS-DOS резервирует память, занятую программой ZERODIV, не позволяя последующим прикладным программам записывать в эту память.

Обработчик прерываний (названный в листинге программы zdiv) получает управление при возникновении прерывания по делению на 0. Обработчик сохраняет все регистры, а затем выводит сообщение с вопросом к пользователю – продолжить или прервать выполнение программы. В рассматриваемом обработчике прерываний можно использовать функции MS-DOS ввода-вывода через консоль, потому что прерывание могло возникнуть, только если выполнялась прикладная программа, и случайные наложения вызовов операционной системы исключены.

Если пользователь, желая продолжить выполнение, вводит C, обработчик просто восстанавливает все регистры и выполняет команду IRET (Возврат из прерывания), возвращая управление прикладной программе. (Естественно, результат операции деления не будет иметь никакой ценности.) Если пользователь, желая прекратить работу, вводит Q, обработчик выходит в MS-DOS. Использование в этом случае Int 21H с функцией 4CH особенно удобно, так как программа может передать код возврата и, кроме того, это единственная функция завершения, которая не использует содержимого каких-либо сегментных регистров.

Примером обработчика внешних (коммуникационный порт) прерываний может служить программа эмуляции терминала TALK, приведенная в гл. 7. Вы можете также еще раз вернуться к обсуждению обработчиков исключительных состояний от нажатия клавиши Ctrl-C и от критических ошибок, проведенному в гл. 5 и 8.

ГЛАВА 14. УСТАНАВЛИВАЕМЫЕ ДРАЙВЕРЫ УСТРОЙСТВ

Драйверы устройств – это модули операционной системы, управляющие аппаратурой. Они изолируют ядро операционной системы от специфических характеристик и особенностей периферийных устройств, подключенных к центральному процессору. Таким образом, взаимоотношения драйвера и ядра напоминают взаимоотношения операционной системы и прикладных программ.

Устанавливаемые драйверы устройств, впервые использованные в MS-DOS версии 2, являются необычайно гибкими средствами. Они позволяют пользователю настраивать компьютер, исходя из конкретных прикладных задач, и конфигурировать его под широкий диапазон периферийных устройств, ограничиваясь при этом минимумом сложных операций и не требуя наложения «заплат» на операционную систему. Даже самый неопытный пользователь может установить в системе новое устройство, вставив плату, скопировав файл с драйвером на загружаемый диск и отредактировав файл конфигурации системы.

Для тех, кто предпочитает программировать самостоятельно, в устанавливаемых драйверах устройств MS-DOS предусмотрен интерфейс с аппаратно-независимым ядром с помощью простой и четко определенной системы функциональных программ и структур данных. Получив необходимую информацию о характеристиках аппаратуры, любой квалифицированный программист на языке ассемблера может успешно включить в сферу действий MS-DOS даже самое причудливое устройство без каких-либо изменений операционной системы и без необходимости вникать в тонкости ее внутреннего построения и детали функционирования.

В ретроспективном плане устанавливаемые драйверы устройств зарекомендовали себя как одно из наиболее полезных средств MS-DOS. И очевидно, что именно они в значительной степени определили быстрое распространение скоростных устройств массовой памяти для ориентированных на MS-DOS машин, сопровождаемое, по законам конкуренции, снижением цен, а также растущее стремление среднего пользователя к самостоятельному усовершенствованию своего компьютера.

Типы драйверов устройств MS-DOS

Драйверы, написанные для MS-DOS, разбиваются на два отчетливо определенных класса:

– драйверы блочно-ориентированных устройств,
– драйверы символьно-ориентированных устройств.

Класс драйвера определяет, какие функции он поддерживает, как он выглядит с точки зрения MS-DOS и каким представляется функционирование соответствующего физического устройства, когда программа выдает запрос на ввод-вывод.

Драйверы символьно-ориентированных устройств

Драйверы символьно-ориентированных устройств управляют периферийными устройствами, выполняющими посимвольный (или побайтовый) ввод и вывод. К таким устройствам относятся, например, терминал и принтер.

Каждый символьно-ориентированный драйвер обычно поддерживает одно устройство. Каждое символьно-ориентированное устройство имеет логическое имя (от

1 до 8 символов), и прикладная программа, используя это имя, может открыть устройство для ввода или вывода, как если бы оно было файлом. Логическое имя – это просто способ идентификации устройства для MS-DOS; оно не имеет физического эквивалента в самом устройстве.

Встроенные в MS-DOS символьно-ориентированные драйверы для консоли, последовательного порта и принтера отличаются тем, что прикладная программа может обращаться к ним тремя различными способами:

- Программа может открыть их по имени (CON, AUX, PRN и т. д.) для ввода или вывода, как любое другое символьно-ориентированное устройство.
- Программа может использовать функциональные вызовы MS-DOS специального назначения (Int 21H с функциями 01–0CH).
- Программа может обратиться к используемым по умолчанию дескрипторам (стандартного ввода, стандартного вывода, стандартной ошибки, стандартного последовательного канала и стандартного принтера), которые не нуждаются в открытии перед использованием.

Число дополнительно устанавливаемых драйверов символьно-ориентированных устройств ограничивается только доступной памятью и необходимостью иметь для каждого драйвера уникальное имя. Если двум или более драйверам присвоено одно и то же логическое имя, последний загруженный драйвер вытеснит все остальные и будет воспринимать все запросы на ввод-вывод, адресуемые к этому логическому имени. Это обстоятельство, однако, иногда можно обратить на пользу дела; например, оно дает возможность заменить используемый по умолчанию драйвер CON, не поддерживающий позиционирования курсора или атрибуты символов, более мощным драйвером ANSI.SYS.

Режим ASCII и двоичный режим

С точки зрения MS-DOS дескриптор, связанный с символьно-ориентированным устройством, может находиться в режиме ASCII или в двоичном режиме. Режим влияет на характер буферизации данных, выполняемой MS-DOS для запросов на чтение или запись. Сам драйвер не осведомлен о режиме, и режим не влияет на функционирование драйвера. Прикладная программа может выбрать режим дескриптора с помощью функции IOCTL (Int 21H с функцией 44H).

В процессе ввода в режиме ASCII MS-DOS запрашивает по одному символу от драйвера и помещает их в собственный внутренний буфер, выводя параллельно на экран (если входным устройством является клавиатура) и проверяя каждый символ на совпадение с Ctrl-C (03H). После того как MS-DOS получает затребованное прикладной программой число символов, или при обнаружении Ctrl-Z, или при нажатии клавиши Enter (в случае ввода с клавиатуры) MS-DOS завершает ввод и копирует данные из своего внутреннего буфера в буфер запрашивающей программы.

Аналогично этому при выводе в режиме ASCII MS-DOS передает драйверу устройства символ за символом и в промежутках проверяет, не введено ли с клавиатуры сочетание Ctrl-C. При обнаружении Ctrl-C MS-DOS прекращает операции ввода или вывода и передает управление программе, адрес которой хранится в векторе прерывания Int 23H.

В двоичном режиме MS-DOS читает или записывает в точности количество байтов, запрошенное прикладной программой, безотносительно к каким-либо управляющим символам вроде Enter или Ctrl-Z. MS-DOS передает через драйвер весь запрос в одной операции, не разбивая его на посимвольные операции чтения или записи, и пересылает символы непосредственно в буфер запрашивающей программы или из него.

Драйверы блочно-ориентированных устройств

Драйверы блочно-ориентированных устройств обычно управляют массовыми запоминающими устройствами прямого доступа, такими, как дисководы гибких или жестких дисков, хотя их можно использовать и для управления устройствами последовательного доступа, например запоминающим устройством на магнитной ленте. Блочно-ориентированные устройства передают данные группами, а не по одному байту за раз. Размер блоков может быть фиксированным (дисководы) или переменным (ленточные устройства).

Блочно-ориентированный драйвер может поддерживать более одного устройства, а также отображать одно физическое устройство на два или более логических. Блочно-ориентированные устройства не имеют логических имен наподобие имен файлов, как это имеет место для символьно-ориентированных устройств. Вместо этого MS-DOS назначает блочно-ориентированным устройствам или логическим дисководам обозначения в алфавитном порядке: A, B и т. д. Каждый логический дисковод содержит файловую систему: блок загрузчика, таблицу размещения файлов, корневой каталог и т. д. (См. гл. 10.)

Позиция драйвера блочно-ориентированного устройства в цепочке драйверов соответствует первой букве, назначенной драйверу. Число логических дисководов, поддерживаемых драйвером, соответствует полному числу назначенных ему букв.

Блочно-ориентированные устройства всегда читают или записывают в точности запрошенное число секторов (если не рассматривать случаи аппаратных ошибок или ошибок адресации) и никоим образом не фильтруют содержимое пересылаемых блоков.

СТРУКТУРА ДРАЙВЕРА УСТРОЙСТВА MS-DOS

Драйвер устройства состоит из трех основных частей (рис. 14.1): заголовок устройства, программа стратегии (strat), программа прерывания (intr).

По ходу этой главы мы обсудим перечисленные части более подробно.

Заголовок устройства

Заголовок устройства (рис. 14.2) расположен в начале драйвера. Заголовок включает связь со следующим драйвером в цепочке, набор флагов атрибутов устройства (рис. 14.3), относительные адреса выполнимых программ стратегии и прерывания, а также имя логического устройства (если устройство символьно-ориентировано, например PRN или COM1) или число логических устройств (если устройство блочно-ориентировано).

Программа стратегии

MS-DOS вызывает программу стратегии (strat), когда драйвер впервые загружается и устанавливается, а потом каждый раз, когда прикладная программа выдает запрос к устройству на ввод-вывод. MS-DOS передает программе стратегии двухсловный указатель структуры данных, называемой заголовком запроса. Эта структура содержит информацию о типе операции. В используемых в настоящее время версиях MS-DOS программа стратегии фактически не выполняет каких-либо операций ввода-вывода, а просто сохраняет указатель заголовка запроса. В программе strat не должно быть каких-либо вызовов Int 21H.

Первые 13 байт заголовка запроса совпадают для всех функций драйвера устройст-

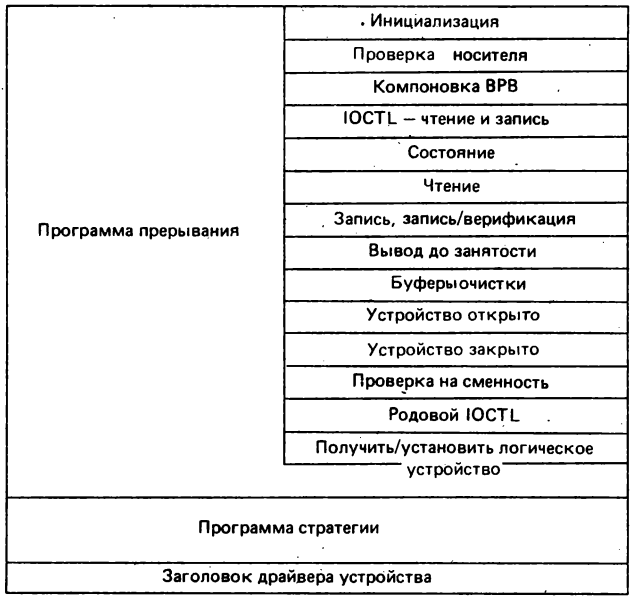


Рис. 14.1. Общая структура устанавливаемого драйвера устройства MS-DOS.

ва и называются поэтому статической частью заголовка. Число и содержимое последующих байт зависят от типа затребованной функции (рис. 14.4). И MS-DOS, и драйвер читают и записывают информацию в заголовок запроса.

Наиболее важным компонентом заголовка запроса является командный код или номер функции, передаваемый в его третьем байте. С помощью командного кода

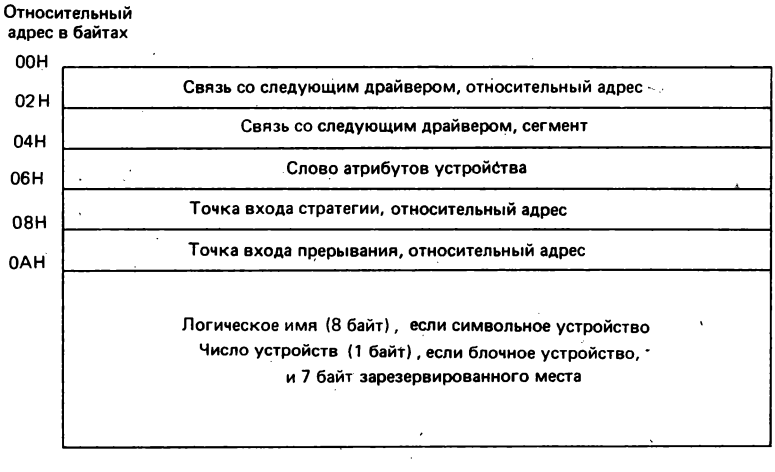


Рис. 14.2. Заголовок драйвера устройства.

Относительные адреса программ strat и intr являются относительными адресами в том же сегменте, который используется для указания на заголовок устройства.

Бит	Значение
15	1, если символьное устройство, 0, если блочное
14	1, если поддерживается IOCTL – чтение и запись
13	*Для блочных устройств:* 1, если для определения характеристик носителя следует использовать BPB в секторе загрузчика; 0, если следует использовать байт идентификации носителя *Для символьных устройств:* 1, если поддерживается вывод до занятости
12	Зарезервирован (должен = 0)
11	1, если поддерживается открыто/закрыто/сменный носитель (MS-DOS версий 3.0 и выше)
7–10	Зарезервированы (должны = 0)
6	1, если поддерживаются родовой IOCTL и получение/установка логического дисковода (MS-DOS версий 3.2 и выше)
5	Зарезервирован (должен = 0)
4	1, если поддерживаются драйвер CON и функция Int 29H быстрого вывода
3	1, если текущее устройство CLOCK$
2	1, если текущее устройство NUL
1	*Для блочных устройств:* 1, если драйвер поддерживает 32-разрядную адресацию секторов (MS-DOS версий 4.0) Для символьных устройств: 1, если стандартное устройство вывода
0	1, если текущее стандартное устройство ввода

Рис. 14.3. Слово атрибутов устройства в заголовке устройства.

В драйверах блочно-ориентированных устройств имеют значение только биты 6, 11 и 13–15 (а также бит 1 в MS-DOS версии 4.0); остальные биты должны быть всегда равны 0.

```
;
; Определение структуры заголовка запроса MS-DOS
;
Request struc                   ; Шаблон структуры заголовка запроса

Rlength     db      ?           ;  0 длина заголовка запроса
Unit        db      ?           ;  1 номер устройства для этого запроса
Command     db      ?           ;  2 командный код заголовка запроса
Status      db      ?           ;  3 слово состояния, возвращаемое драйвером
Reserve     db      8 dup (?)   ;  5 зарезервированная область
Media       db      ?           ; 13 байт описания носителя
Address     dd      ?           ; 14 адрес памяти для пересылки
Count       dw      ?           ; 18 значение счетчика байт/сектор
Sector      dw      ?           ; 20 номер начального сектора

Request     ends                ; Конец шаблона структуры заголовка запроса
```

Рис. 14.4. Формат заголовка запроса.

Только первые 13 байт являются общими для всех функций драйвера; число и определения последующих байтов зависят от типа функции. Показанная здесь структура используется подфункциями чтения и записи.

выбирается подфункция драйвера: чтение, запись или состояние. Через заголовок запроса драйверу передаются также номер устройства, адрес пересылки и счетчик секторов или байтов.

Программа прерывания

Последней и самой важной частью драйвера устройства является программа прерывания (intr), которую MS-DOS вызывает сразу вслед за вызовом программы стратегии. Программа прерывания воплощает само существо драйвера устройства; она выполняет (сама или с помощью других резидентных программ) фактические операции ввода или вывода в соответствии с информацией, переданной в заголовке запроса. Программа intr не должна выполнять вызовов каких-либо функций Int 21H, за исключением ограниченного их набора в процессе инициализации драйвера.

После завершения операции ввода-вывода программа прерывания с помощью поля состояния в заголовке запроса информирует ядро DOS о результате выполнения затребованной операции ввода-вывода. Программа прерывания может использовать и другие поля в заголовке запроса для передачи назад такой полезной информации, как число фактически переданных секторов или байтов.

Программа прерывания обычно состоит из следующих элементов:

- Набор подпрограмм для реализации функций различных типов, которые может запросить MS-DOS (иногда эти подпрограммы называют программами командных кодов).
- Централизованная точка входа, где сохраняются все используемые регистры, из заголовка запроса извлекается код запрашиваемой функции и осуществляется переход

Бит(ы)	Значение
15	Ошибка
12–14	Зарезервированы
9	Занято
8	Выполнено
0–7	Код ошибки, если бит 15 = 1

Рис. 14.5. Значения слова возвращаемого состояния в заголовке запроса.

Код	Значение
0	Нарушение защиты записи
1	Неизвестное устройство
2	Дисковод не готов
3	Неизвестная команда
4	Ошибка данных (CRC)
5	Неверная длина структуры запроса
6	Ошибка поиска
7	Неизвестный носитель
8	Сектор не найден
9	В принтере нет бумаги
0AH	Ошибка записи
0BH	Ошибка чтения
0CH	Общая ошибка
0D–0EH	Зарезервированы
0FH	Неправильная смена диска (MS-DOS версий 3.0 и выше)

Рис. 14.6. Коды ошибок драйвера, возвращаемые в битах 0–7 слова возвращаемого состояния в заголовке запроса.

на соответствующую программу командного кода (обычно с помощью таблицы переходов).

● Централизованная точка выхода, где в заголовок запроса заносятся коды состояния и ошибки (рис. 14.5 и 14.6) и восстанавливается прежнее содержимое используемых регистров.

Программы командных кодов, реализующие функции, поддерживаемые устанавливаемым драйвером устройства, детально обсуждаются ниже.

Несмотря на свое название, программа прерывания никогда не инициируется асинхронно (например, по прерыванию завершения ввода-вывода). Таким образом, разделение функций между программами стратегии и прерывания, во всяком случае в используемых в настоящее время версиях MS-DOS, носит искусственный характер.

ПРОГРАММЫ КОМАНДНЫХ КОДОВ

Для драйверов устройств MS-DOS определено всего 20 командных кодов. Ниже приведены командные коды (идущие не подряд), названия соответствующих программ прерываний для драйверов и версии MS-DOS, в которых были введены эти программы.

Командный код	Функция	Символьный драйвер	Блочный драйвер	Версия
0	Init (инициализация)	×	×	2.0
1	Проверка носителя		×	2.0
2	Построение BPB		×	2.0
3	IOCTL-чтение	×	×	2.0
4	Чтение	×	×	2.0
5	Неразрушающее чтение	×		2.0
6	Состояние ввода	×		2.0
7	Очистка буферов ввода	×		2.0
8	Запись	×	×	2.0
9	Запись с верификацией		×	2.0
10	Состояние вывода	×		2.0
11	Очистка буферов вывода	×		2.0
12	IOCTL-запись	×	×	2.0
13	Устройство открыто	×	×	3.0
14	Устройство закрыто	×	×	3.0
15	Сменный носитель		×	3.0
16	Вывод до занятости	×		3.0
19	Родовой IOCTL	×	×	3.2
23	Получение логического устройства		×	3.2
24	Установка логического устройства		×	3.2

Как видно из приведенной таблицы, секция прерывания драйвера должна для всех версий MS-DOS поддерживать функции с 0 до 12. Драйверы, предназначенные для MS-DOS версии 3.0, дополнительно поддерживают (не обязательно) еще четыре функции, а драйверы MS-DOS для версий 3.2 и выше могут поддерживать еще на три больше (всего получается 20). MS-DOS проверяет биты слова атрибутов в заголовке драйвера устройства, чтобы определить, поддерживает ли драйвер необязательные функции, и если поддерживает, то какие.

Некоторые функции относятся только к драйверам символьно-ориентированных устройств, некоторые – только к драйверам блочно-ориентированных устройств; отдельные функции используются и в тех и в других драйверах. В любом случае драйверы обоих типов должны для каждой функции иметь выполнимую программу, даже если она не делает ничего, кроме установки флага «выполнено» в слове состояния заголовка запроса.

В приведенном ниже описании командных кодов, обозначение RH (от request header, заголовок запроса) относится к заголовку запроса, адрес которого был передан программе стратегии в регистрах ES:BX; BYTE обозначает 8-разрядный параметр, WORD – 16-разрядный параметр, а DWORD – дальний указатель (16-разрядный относительный адрес, за которым следует 16-разрядный сегмент).

Функция 00H (0): Инициализация драйвера

MS-DOS вызывает функцию инициализации драйвера (init) только однажды, при первой загрузке драйвера. Эта функция выполняет всю необходимую инициализацию устройства, заполнение векторов прерываний и т. д. Программа инициализации должна вернуть адрес начала свободной памяти после программы драйвера (адрес разрыва), так что MS-DOS знает, где можно начинать строить определенные управляющие структуры, а затем загружать следующий устанавливаемый драйвер. Если драйвер предназначен для блочно-ориентированного устройства, программа инициализации должна также вернуть число устройств и адрес массива указателей BPB.

MS-DOS использует число устройств, возвращаемое блочно-ориентированным драйвером в заголовке запроса, для назначения идентификаторов дисководов. Если, например, текущее старшее имя драйвера есть D, а инициализируемый драйвер поддерживает четыре устройства, MS-DOS назначит им буквы E, F, G и H. Хотя заголовок драйвера устройства тоже имеет поле для числа устройств, MS-DOS не анализирует его содержимое.

Массив указателей BPB представляет собой массив относительных адресов в блоке параметров BIOS (рис. 14.7). Каждое устройство, определенное драйвером, должно иметь запись в этом массиве, хотя все эти записи могут указывать на один и тот же BPB ради экономии памяти. В процессе выполнения загрузки операционной системы MS-DOS просматривает все BPB, описываемые всеми устройствами во всех драйверах блочно-ориентированных устройств, чтобы определить максимальный размер сектора на устройствах системы, и использует затем эту информацию при установке размера своего кэш-буфера.

Средства операционной системы, которые могут использоваться программой инициализации во время загрузки, весьма ограниченны – только прерывание Int 21H с функциями от 01H до 0CH и 30H. Этого достаточно для проверки номера версии MS-DOS и вывода идентификатора драйвера или сообщения об ошибке.

Байт(ы)	Содержимое
00–01H	Число байтов в секторе
02H	Число секторов в единице выделения (степень двух)
03H–04H	Число зарезервированных секторов (начиная с сектора 0)
05H	Число таблиц размещения файлов (FAT)
06H–07H	Максимальное число записей в корневом каталоге
08H–09H	Полное число секторов на носителе
0AH	Байт дескриптора носителя
0BH–0CH	Число секторов в одной FAT
0DH–0EN	Число секторов на дорожке (версий 3.0 и выше)
0FH–1OH	Число головок (версий 3.0 и выше)
11H–12H	Число скрытых секторов (версий 3.0 и выше)
13H–14H	Старшее слово числа скрытых секторов (версия 4.0)
15H–18H	Если байты 8 – 9 = 0, полное число секторов на носителе (версия 4.0)
19H–1EH	Зарезервированы, должны = 0 (версия 4.0)

Рис. 14.7. Структура блока параметров BIOS (BPB).

Каждый форматированный диск содержит копию своего BPB в секторе загрузчика (см. гл. 10).

Многие программисты помещают программу инициализации в конец драйвера и возвращают этот адрес как адрес начала свободной памяти, чтобы MS-DOS могла использовать память, занимаемую программой инициализации, после того, как эта программа закончила свою работу. Если программа инициализации обнаруживает, что устройство отсутствует или неисправно, и требуется полностью прекратить установку драйвера, чтобы он не занимал память, программа должна вернуть нуль в качестве числа устройств и установить адрес свободной памяти, равный CS:0000H. (Драйвер символьно-ориентированного устройства для прекращения установки должен очистить бит 15 слова атрибутов в заголовке драйвера, а затем установить поле устройств и адрес свободной памяти, как если бы это был драйвер блочно-ориентированного устройства.)

При вызове функции инициализации:

RH + 2	BYTE	Командный код = 0
RH + 18	DWORD	Указатель символа, следующего за знаком равенства в строке CONFIG.SYS загрузки драйвера (эта информация только для чтения)
RH + 22	BYTE	Номер дисковода для первого устройства этого блочно-ориентированного драйвера (0 = A, 1 = B и т.д.) (Только MS-DOS версии 3.)

Функция возвращает:

RH + 3	WORD	Состояние
RH + 13	BYTE	Число устройств (только для блочных устройств)
RH + 14	DWORD	Адрес начала свободной памяти за драйвером (адрес разрыва)
RH + 18	DWORD	Массив указателей BPB (только для блочно-ориентированных устройств)

Функция 01H (1): Проверка носителя

Функция проверки носителя имеет смысл только для блочно-ориентированных устройств; в драйверах символьно-ориентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено». Функция вызывается в том случае, если повисает запрос к дисководу, отличающийся от простых операций чтения или записи файла. MS-DOS передает функции байт-описания носителя для того диска, который с точки зрения DOS установлен на дисководе (рис. 14.8). Программа проверки носителя в случае успешного выполнения возвращает код, указывающий на то, сменился ли диск с момента последней передачи данных. Если программа проверки

Код	Значение
0F0H	3,5″, 2 стороны, 18 секторов
0F8H	Жесткий диск
0F9H	3,5″, 2 стороны, 9 секторов
0F9H	5,25″, 2 стороны, 15 секторов
0FCH	5,25″, 1 сторона, 9 секторов
0FDH	5,25″, 2 стороны, 9 секторов
0FEH	5,25″, 1 сторона, 8 секторов
0FFH	5,25″, 2 стороны, 8 секторов

Рис. 14.8. Коды, используемые в настоящее время в байте описания носителя в заголовке запроса в предположении, что бит 13 в слове атрибутов заголовка драйвера равен 0.

носителя удостоверяет, что смены диска не было, MS-DOS может опустить повторное чтение FAT перед обращением к каталогу, что повышает общую производительность.

MS-DOS реагирует следующим образом на результаты выполнения функции проверки носителя:

- Если смены диска не было, MS-DOS продолжает операции доступа к диску.
- Если диск сменили, MS-DOS делает недействительными все буферы, связанные с этим устройством, включая буферы с ожидающими записи данными (эти данные просто теряются), выполняет вызов функции построения BPB, после чего читает FAT диска и его каталог.
- Если состояние смены диска неизвестно, действия MS-DOS зависят от состояния ее внутренних буферов. Если в буферах находятся ожидающие записи данные, MS-DOS предполагает, что смены диска не было, и записывает данные (рискуя повредить файловую структуру нового диска, если в действительноети диск сменили). Если же буферы пусты или уже были сброшены на диск, MS-DOS предполагает, что диск сменили, и действует так же, как и в случае получения кода смены диска.

Если установлен бит 11 слова атрибутов заголовка устройства (т.е. драйвер поддерживает необязательные функции открыто/закрыто/сменный носитель), на машине установлена система MS-DOS версии 3.0 или более поздней, и функция возвращает код смены диска (−1), функция должна также вернуть сегмент и относительный адрес строки ASCIIZ метки тома для предыдущего диска на дисководе. (Если драйвер не содержит метки тома, он может вернуть указатель строки ASCIIZ NO NAME, Имя отсутствует.) Если MS-DOS определяет, что диск был сменен при наличии в его буферах незаписанных данных, выдается критическая ошибка 0FH (Неправильная смена диска). Прикладная программа может уловить эту критическую ошибку и предложить пользователю установить предыдущий диск.

При вызове функции проверки носителя:

RH + 1	BYTE	Код устройства
RH + 2	BYTE	Командный код = 1
RH + 13	BYTE	Байт описания носителя

Функция возвращает:

RH + 3	WORD	Состояние
RH + 14	BYTE	Код смены носителя: −1, если диск сменили 0, если неизвестно, была ли смена диска 1, если смены не было
RH + 15	DWORD	Указатель на метку предыдущего тома, если бит 11 атрибутов устройства = 1 и диск сменили (MS-DOS версий 3.0 и более поздних)

Функция 02H (2): Построение блока параметров (BPB) BIOS

Функция построения BPB имеет смысл только для блочно-ориентированных устройств; в драйверах символьно-ориентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено». Ядро использует эту функцию, чтобы получить указатель допустимого BPB (см. рис. 14.7) для текущего диска, и вызывает ее, если программой проверки носителя возвращен код смены диска или возвращен код «неизвестно, была ли смена диска» и отсутствуют «грязные»

буферы (буферы с измененными данными, еще не записанными на диск). Таким образом, вызов этой функции указывает, что была выполнена правильная смена диска.

Функция построения BPB получает указатель односекторного буфера в заголовке запроса. Если бит 13 в слове атрибутов заголовка драйвера равен 0, буфер содержит первый сектор FAT (включающий байт описания носителя) и не должен изменяться драйвером. Если бит 13 установлен, драйвер может использовать буфер как рабочую область.

При вызове функции построения BPB:

RH + 1	BYTE	Код устройства
RH + 2	BYTE	Командный код = 2
RH + 13	BYTE	Байт описания носителя
RH + 14	DWORD	Адрес буфера

Функция возвращает:

RH + 3	WORD	Состояние
RH + 18	DWORD	Указатель нового BPB

При использовании MS-DOS версий 3.0 и более поздних, если установлен бит 11 в слове атрибутов заголовка устройства, эта программа должна также считать с диска и сохранить метку тома.

Функция 03H (3): Чтение управляющих данных ввода-вывода (IOCTL-чтение)

Функция IOCTL-чтения дает возможность драйверу устройства передавать управляющую информацию непосредственно в прикладную программу. Эта функция вызывается, только если в слове атрибутов устройства установлен бит 14. При IOCTL-вызовах MS-DOS не выполняет контроля ошибок.

При вызове функции IOCTL-чтения:

RH + 1	BYTE	Код устройства (блочно-ориентированные устройства)
RH + 2	BYTE	Командный код = 3
RH + 13	BYTE	Байт описания носителя
RH + 14	DWORD	Адрес пересылки
RH + 18	WORD	Счетчик байтов/секторов
RH + 20	WORD	Номер начального сектора (блочно-ориентированные устройства)

Функция возвращает:

RH + 3	WORD	Состояние
RH + 18	WORD	Число фактически переданных байтов или секторов

Функция 04H (4): Чтение

Функция чтения пересылает данные из устройства в указанный буфер памяти. Если в процессе чтения возникает ошибка, функция должна установить состояние ошибки и, кроме того, сообщить число успешно переданных байтов или секторов; просто сообщить об ошибке недостаточно.

При вызове функции чтения:

RH + 1	BYTE	Код устройства (блочно-ориентированные устройства)
RH + 2	BYTE	Командный код = 4
RH + 13	BYTE	Байт описания носителя
RH + 14	DWORD	Адрес пересылки
RH + 18	WORD	Счетчик байтов/секторов
RH + 20	WORD	Номер начального сектора (блочно-ориентированные устройства)

Для операций чтения с блочно-ориентированных устройств в MS-DOS версии 4, если логическое устройство больше 32 Мбайт и бит 1 слова атрибутов драйвера установлен, используется другая структура вызова:

RH + 1	BYTE	Код устройства
RH + 2	BYTE	Командный код = 4
RH + 13	BYTE	Байт описания носителя
RH + 14	DWORD	Адрес пересылки
RH + 18	WORD	Счетчик секторов
RH + 20	WORD	Содержит −1 как флаг использования 32-разрядного номера сектора
RH + 26	DWORD	32-разрядный номер начального сектора

Функция возвращает:

RH + 3	WORD	Состояние
RH + 18	WORD	Число фактически переданных байтов или секторов
RH + 22	DWORD	Если возвращена ошибка 0FH, указатель метки тома (MS-DOS версий 3.0 и выше)

При использовании MS-DOS версий 3.0 и выше эта программа определяет, не был ли диск неправильно сменен, с помощью счетчика открытых файлов, поддерживаемого функциями «открыто» и «закрыто» (0DH и 0EH), и байта описания носителя.

Функция 05H (5): Неразрушающее чтение

Функция неразрушающего чтения имеет смысл только для символьно-ориентированных устройств; в драйверах блочно-ориентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено». Функция возвращает следующий символ, который был бы получен функцией чтения (командный код 4), однако не извлекает этот символ из внутреннего буфера драйвера. MS-DOS использует эту функцию в процессе выполнения других операций для проверки драйвера консоли на висящий символ Ctrl-C.

При вызове функции неразрушающего чтения:

RH + 2	BYTE	Командный код = 5

Функция возвращает:

RH + 3	WORD	Состояние Если бит занятости = 0, по крайней мере один символ ожидает

Если бит занятости = 1, нет ожидающих символов
RH + 13 BYTE Символ (если бит занятости = 0)

Функция 06H (6): Состояние ввода

Функция состояния ввода имет смысл только для символьно-ориентированных устройств; в драйверах блочно-ориентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено». Эта функция возвращает текущее состояние ввода устройства, позволяя MS-DOS определить, имеются ли ожидающие символы в буфере упреждающего ввода. Если символьно-ориентированное устройство не имеет буфера упреждающего ввода, программа состояния ввода должна во всех случаях возвращать бит занятости сброшенным, чтобы MS-DOS не ожидала бесконечно долго перед вызовом функции чтения (04H) или неразрушающего чтения (05H).

При вызове функции состояния ввода:

RH + 2 BYTE Командный код = 6

Функция возвращает:

RH + 3 WORD Состояние:
Если бит занятости = 1, запрос на чтение направляется в физическое устройство
Если бит занятости = 0, символы уже в буфере устройства и запрос на чтение сразу же выполняется

Функция 07H (7): Очистка буферов ввода

Функция очистки буферов ввода имеет смысл только для символьно-ориентированных устройств; в драйверах блочно-ориентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено». Эта функция приводит к отбрасыванию всех данных, ожидающих в буфере ввода.

При вызове функции очистки буферов ввода:

RH + 2 BYTE Командный код = 7

Функция возвращает:

RH + 3 WORD Состояние

Функция 08H (8): Запись

Функция записи пересылает данные из указанного буфера памяти в устройство. Если в процессе записи возникает ошибка, функция записи должна установить состояние ошибки и, кроме того, сообщить число успешно переданных байтов или секторов; просто сообщить об ошибке недостаточно.

При вызове функции записи:

RH + 1 BYTE Код устройства (блочно-ориентированные устройства)

RH + 2	BYTE	Командный код = 8
RH + 13	BYTE	Байт описания носителя
RH + 14	DWORD	Адрес пересылки
RH + 18	WORD	Счетчик байтов/секторов
RH + 20	WORD	Номер начального сектора (блочно-ориентированные устройства)

Для операций записи на блочно-ориентированные устройства в MS-DOS версии 4, если логическое устройство больше 32 Мбайт и бит 1 слова атрибутов драйвера установлен, используется другая структура запроса:

RH + 1	BYTE	Код устройства
RH + 2	BYTE	Командный код = 8
RH + 13	BYTE	Байт описания носителя
RH + 14	DWORD	Адрес пересылки
RH + 18	WORD	Счетчик секторов
RH + 20	WORD	Содержит −1 как флаг использования 32-разрядного номера сектора
RH + 26	DWORD	32-разрядный номер начального сектора

Функция возвращает:

RH + 3	WORD	Состояние
RH + 18	WORD	Число фактически переданных байтов или секторов
RH + 22	DWORD	Если возвращена ошибка 0FH, указатель метки тома (MS-DOS версий 3.0 и выше)

При использовании MS-DOS версий 3.0 и выше эта программа определяет, не был ли диск неправильно сменен, с помощью счетчика открытых файлов, поддерживаемого функциями «открыто» и «закрыто» (0DH и 0EH), и байта описания носителя.

Функция 09H (9): Запись с верификацией

Функция записи с верификацией пересылает данные из указанного буфера памяти в устройство. В тех случаях, когда это возможно, функция выполняет проверку записанных данных путем их повторного чтения после записи. В остальных отношениях функция 09H не отличается от функции 08H.

Функция 0AH (10): Состояние вывода

Функция состояния вывода имеет смысл только для символьно-ориентированных устройств; в драйверах блочно-орентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено». Эта функция возвращает текущее состояние вывода устройства.

При вызове функции состояния вывода:

RH + 2	BYTE	Командный код = 10 (0AH)

Функция возвращает:

RH + 3	WORD	Состояние:

Если бит занятости = 1, запрос на запись ждет завершения текущего запроса
Если бит занятости = 0, устройство свободно и запрос на запись реализуется немедленно

Функция 0BH (11): Очистка буферов вывода

Функция очистки буферов вывода имеет смысл только для символьно-ориентированных устройств; в драйверах блочно-ориентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено». Эта функция очищает буфер вывода (если таковой существует) и отбрасывает все висящие запросы на вывод.

При вызове функции очистки буфера вывода:

RH + 2	BYTE	Командный код = 11 (0BH)

Функция возвращает:

RH + 3	WORD	Состояние

Функция 0CH (12): Запись управляющих данных ввода-вывода (IOCTL-запись)

Функция IOCTL-записи дает возможность прикладной программе передавать управляющие данные непосредственно драйверу. При IOCTL-вызовах MS-DOS не выполняет контроля ошибок.

При вызове функции IOCTL-записи:

RH + 1	BYTE	Код устройства (блочно-ориентированные устройства)
RH + 2	BYTE	Командный код = 12 (0CH)
RH + 13	BYTE	Байт описания носителя
RH + 14	DWORD	Адрес пересылки
RH + 18	WORD	Счетчик байтов/секторов
RH + 20	WORD	Номер начального сектора (блочно-ориентированные устройства)

Функция возвращает:

RH + 3	WORD	Состояние
RH + 18	WORD	Число фактически переданных байтов или секторов

Функция 0DH (13): Устройство открыто

Функция «устройство открыто» поддерживается только версиями MS-DOS 3.0 и выше и вызывается, только если установлен бит 11 слова атрибутов устройства в заголовке драйвера.

С блочно-ориентированными устройствами функцию «устройство открыто» можно использовать для управления локальной буферизацией и для увеличения значения счетчика числа открытых файлов на устройстве. Однако при этом следует соблюдать осторожность, потому что программа, осуществляющая доступ к файлам с помощью

FCB, часто оказывается не в состоянии закрыть их, в результате чего содержимое счетчика открытых файлов теряет смысл. В качестве меры предосторожности можно предложить при получении положительного ответа на запрос о смене носителя и последующем вызове функции построения BPB сбрасывать в нуль счетчик открытых файлов, не очищая при этом буфера.

С символьно-ориентированными устройствами функцию «устройство открыто» можно использовать для пересылки инициализирующей устройство строки (ее можно установить в драйвере прикладной программой с помощью функции IOCTL-записи) или для запрета одновременного обращения к символьно-ориентированному устройству нескольких процессов. Заметьте, что определенные заранее дескрипторы устройств CON, AUX и PRN всегда открыты.

При вызове функции «устройство открыто»:

RH + 1	BYTE	Код устройства (блочно-ориентированные устройства)
RH + 2	BYTE	Командный код = 13 (0DH)

Функция возвращает:

RH + 3	WORD	Состояние

Функция 0EH (14): Устройство закрыто

Функция «устройство закрыто» поддерживается только версиями MS-DOS 3.0 и выше и вызывается, только если установлен бит 11 слова атрибутов устройства в заголовке драйвера.

С блочно-ориентированными устройствами функцию «устройство закрыто» можно использовать для управления локальной буферизацией и для уменьшения значения счетчика числа открытых файлов на устройстве; если значение счетчика доходит до 0, все файлы закрыты и драйвер должен очистить буферы, так как пользователь может сменить диски.

С символьно-ориентированными устройствами функцию «устройство закрыто» можно использовать для пересылки аппаратно-зависимой строки, завершающей ввод-вывод, например для перевода страницы. (Эту строку можно установить в драйвере прикладной программой с помощью функции IOCTL-записи.)

При вызове функции «устройство закрыто»:

RH + 1	BYTE	Номер устройства (блочно-ориентированные устройства)
RH + 2	BYTE	Командный код = 14 (EH)

Функция возвращает:

RH + 3	WORD	Состояние

Функция 0FH (15): Сменный носитель

Функция «сменный носитель» поддерживается только версиями MS-DOS 3.0 и выше и только на блочно-ориентированных устройствах; в драйверах символьно-ориентированных устройств функция не должна выполнять никаких действий, кроме установки флага «выполнено». Эта функция вызывается, только если установлен бит 11 слова атрибутов устройства в заголовке драйвера.

При вызове функции «сменный носитель»:

RH + 1	BYTE	Код устройства
RH + 2	BYTE	Командный код = 15 (0FH)

Функция возвращает:

RH + 3	WORD	Состояние: Если бит занятости = 1, носитель несменный Если бит занятости = 0, носитель сменный

Функция 10H (16): Вывод до занятости

Функция вывода до занятости поддерживается только версиями MS-DOS 3.0 и выше и только на символьно-ориентированных устройствах; в драйверах блочно-ориентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено». Эта функция пересылает данные из указанного буфера памяти к устройству, продолжая пересылку до занятости устройства. Функция вызывается, только если установлен бит 13 слова атрибутов устройства в заголовке драйвера.

Функция носит оптимизационный характер и включена специально для использования спулерами принтеров. Для функции не является ошибкой, если возвращаемое ею число переданных байтов меньше затребованного.

При вызове функции вывода до занятости:

RH + 2	BYTE	Командный код = 16 (10H)
RH + 14	DWORD	Адрес пересылки
RH + 18	WORD	Счетчик байтов

Функция возвращает:

RH + 3	WORD	Состояние
RH + 18	WORD	Число фактически переданных байтов

Функция 13H (19): Родовой IOCTL

Функция родового IOCTL поддерживается только версиями MS-DOS 3.2 и выше и вызывается, только если установлен бит 6 слова атрибутов устройства в заголовке драйвера. Функция соответствует средствам MS-DOS для обслуживания родового IOCTL, предоставляемым прикладным программам через прерывание Int 21H с функцией 44H и подфункциями 0CH и 0DH.

Функции родового IOCTL передаются коды категории (главный) и функции (дополнительный), содержимое регистров SI и DI в точке вызова IOCTL, а также сегмент и относительный адрес буфера данных. Этот буфер в свою очередь содержит другую информацию, формат которой определяется главным и дополнительным кодами, передаваемыми в заголовке запроса. Драйвер должен интерпретировать главный и дополнительный коды в заголовке запроса вместе с содержимым дополнительного буфера с целью определения требуемой от него операции, затем установить флаг «выполнено» в слове состояния заголовка запроса и возвратить соответствующую информацию в заголовке запроса и буфере данных.

Функции, выполняемые запросом родового IOCTL (если их поддерживает драйвер), включают настройку драйвера для нестандартных форматов диска, чтение и запись целых дорожек диска, а также форматирование и верификацию дорожек. Функция родового IOCTL сделана открытой, чтобы с ее помощью можно было легко расширять определение драйвера устройства для работы с будущими версиями MS-DOS.

При вызове функции родового IOCTL:

RH + 1	BYTE	Номер устройства (блочно-ориентированные устройства)
RH + 2	BYTE	Командный код = 19 (13H)
RH + 13	BYTE	Код категории (главный)
RH + 14	BYTE	Код функции (дополнительный)
RH + 15	WORD	Содержимое регистра SI
RH + 17	WORD	Содержимое регистра DI
RH + 19	DWORD	Адрес пакета данных родового IOCTL

Функция возвращает:

RH + 3	WORD	Состояние

Функция 17H (23): Получение логического устройства

Функция получения логического устройства поддерживается только версиями MS-DOS 3.2 и выше и только на блочно-ориентированных устройствах; в драйверах символьно-ориентированных устройств она не должна выполнять никаких действий, кроме установки флага «выполнено» в слове состояния. Эта функция вызывается, только если установлен бит 6 слова атрибутов устройства в заголовке устройства. Функция соответствует средствам MS-DOS для получения отображения логического устройства, предоставляемым прикладным программам через прерывание Int 21H с функцией 44H и подфункцией 0EH.

Функция получения логического устройства возвращает код последней буквы, используемой для ссылок на данное устройство; если устройству назначена только одна буква в качестве имени дисковода, возвращаемый код устройства должен быть равен 0. Таким образом, с помощью этой функции можно определить, присвоена ли данному устройству в качестве имени дисковода только одна или несколько букв.

При вызове функции получения логического устройства:

RH + 1	BYTE	Код устройства
RH + 2	BYTE	Командный код = 23 (17H)

Функция возвращает:

RH + 1	BYTE	Код последней буквы, используемой для ссылок на устройство
RH + 3	WORD	Состояние

Функция 18H (24): Установка логического устройства

Функция установки логического устройства поддерживается только версиями MS-DOS 3.2 и выше и только на блочно-ориентированных устройствах; в драйверах символьно-ориентированных устройств она не должна выполнять никаких действий,

кроме установки флага «выполнено» в слове состояния. Эта функция вызывается, только если установлен бит 6 слова атрибутов устройства в заголовке устройства. Функция соответствует средствам MS-DOS для установки отображения логического устройства, предоставляемым прикладным программам через прерывание Int 21H с функцией 44H и подфункцией 0FH.

Функция установки логического устройства информирует драйвер о следующем идентификаторе логического дисковода, который будет использован для ссылки на данное физическое устройство. Код устройства, передаваемый в этом случае ядром MS-DOS, характеризует номер логического дисковода среди логических дисководов, поддерживаемых этим конкретным драйвером, причем нумерация начинается с нуля. Если, например, драйвер поддерживает два устройства-дисковода гибких дисков (A и B), только один физический дисковод гибких дисков существует в системе, а функция установки логического устройства вызывается с номером устройства, равным 1, то драйвер информируется о том, что следующий запрос ядра на чтение или запись будет адресован к дисководу B.

При вызове функции установки логического устройства:

RH + 1	BYTE	Код устройства
RH + 2	BYTE	Командный код = 24 (18H)

Функция возвращает:

RH + 3	WORD	Состояние

ОБРАБОТКА ТИПИЧНОГО ЗАПРОСА НА ВВОД-ВЫВОД

Для запроса прикладной программой от MS-DOS операции ввода-вывода следует загрузить регистры соответствующими значениями и вызвать прерывание Int 21H. Это приводит к последовательности следующих действий:

1. MS-DOS просматривает свои внутренние таблицы и определяет, какой драйвер устройства должен получить запрос на ввод-вывод.
2. MS-DOS создает пакет данных заголовка запроса в зарезервированной области памяти. (Для дисковых запросов ввода-вывода MS-DOS после анализа каталогов диска и таблицы размещения файлов преобразует информацию о файлах и записях в номера логических секторов.)
3. MS-DOS передает управление на входную точку strat в драйвере устройства, передавая адрес заголовка запроса в регистрах ES:BX.
4. Драйвер устройства сохраняет адрес заголовка запроса в локальной переменной и выполняет команду дальнего возврата (FAR RETURN).
5. MS-DOS передает управление на входную точку intr в драйвере устройства.
6. Программа прерывания сохраняет все регистры, находит адрес заголовка запроса, сохраненный программой стратегии, извлекает код функции и осуществляет переход на соответствующую подпрограмму командного кода для выполнения функции.
7. Если была затребована пересылка данных на блочно-ориентированном устройстве, подпрограмма чтения или записи в драйвере преобразует номер логического сектора в номера головки и дорожки и в адрес физического сектора на закрепленном устройстве, а затем выполняет операцию ввода-вывода. Поскольку в одном заголовке запроса может быть затребована многосекторная передача, один запрос MS-DOS к драйверу может привести к многим командам чтения или записи, адресованным контроллеру диска.

8. После завершения затребованной функции программа прерывания устанавливает слово состояния и другую информацию в заголовке запроса, восстанавливает содержимое всех регистров, соответствующее точке входа, и выполняет команду дальнего возврата (FAR RETURN).

9. MS-DOS пребразует состояние возврата драйвера в код возврата и состояние флага переноса, соответствующие затребованной функции MS-DOS Int 21H, и возвращает управление прикладной программе.

Обратите внимание на то, что один запрос из прикладной программы может привести к передаче от MS-DOS к драйверу многих заголовков запроса. Например, при попытке открыть файл в подкаталоге ранее недоступного дисковода могут потребоваться следующие действия:

– чтение сектора загрузчика диска с целью получения BPB;

– чтение одного, нескольких или многих секторов корневого каталога с целью нахождения записи, соответствующей подкаталогу, и получения номера его начального кластера;

– чтение одного, нескольких или многих секторов таблицы размещения файлов и самого подкаталога с целью нахождения записи, соответствующей требуемому файлу.

ДРАЙВЕР CLOCK: ОСОБЫЙ СЛУЧАЙ

MS-DOS использует устройство CLOCK для занесения в блоки управления файлами и записи каталогов отметок даты и времени, а также для предоставления средств работы с датой и временем прикладным программам. Это устройство характеризуется необычным типом взаимодействия с MS-DOS – из драйвера читается или в драйвер записывается 6-байт последовательность; соответственно драйвер получает или устанавливает текущие дату и время. Последовательность имеет следующий формат:

0	1	2	3	4	5
Дни, младший байт	Дни, старший байт	Минуты	Часы	Секунды/100	Секунды

Значение, передаваемое в поле для дней, представляет собой 16-разрядное целое, характеризующее число дней, истекших с 1 января 1980 г.

Драйвер часов CLOCK может иметь любое имя логического устройства, потому что MS-DOS для идентификации устройства использует не его имя, а бит CLOCK слова атрибутов устройства в заголовке устройства. В системах IBM PC устройство «часы» имеет имя логического устройства CLOCK$.

НАПИСАНИЕ И УСТАНОВКА ДРАЙВЕРА УСТРОЙСТВА

Теперь, после обсуждения структуры и возможностей устанавливаемых драйверов устройств, предназначенных для среды MS-DOS, мы можем перейти к рассмотрению механических шагов их ассемблирования и компоновки.

Ассемблирование

Драйверы устройств для MS-DOS всегда имеют базовый адрес 0, но в остальных отношениях ассемблируются, компонуются и преобразуются в выполнимые модули точно так же, как файлы .COM. (Хотя MS-DOS всегда может загрузить устанавливаемый драйвер в формате файла .EXE, однако использование формата .EXE неоправданно увеличивает сложности написания и отладки драйверов, не давая при этом

ощутимых преимуществ. К тому же, драйверы в формате .EXE нельзя использовать с некоторыми версиями MS-DOS разработки фирмы IBM, потому что загрузчик файлов .EXE расположен в программе COMMAND.COM, которая отсутствует в момент загрузки устанавливаемых драйверов.) Драйвер не должен содержать объявленного сегмента стека; кроме того, при написании драйвера следует соблюдать другие ограничения, относящиеся к программам, представляющим образ памяти (.COM) и описанным в гл. 3. Драйвер может быть загружен в любое место памяти, так что

```
        name    driver
        page    55,132
        title   DRIVER.ASM Device-Driver Skeleton
                        ;Структура драйвера устройства

;
; DRIVER.ASM  Функциональная схема драйвера устройства MS-DOS
;
; Программы командных кодов, включенные в драйвер, представляют
; собой просто заглушки и не оказывают никаких действий, кроме
; возврата состояния "выполнено" при отсутствии ошибки
;
; Авторские права 1988 Рей Данкан
;

_TEXT   segment word public 'CODE'

        assume  cs:_TEXT,ds:_TEXT,es:NOTHING

        org     0

MaxCmd  equ     24              ; Максимально допустимый командный код:
                                ; 12 для MS-DOS 2
                                ; 16 для MS-DOS 3.0 - 3.1
                                ; 24 для MS-DOS 3.2 - 3.3
cr      equ     0dh             ; Код ASCII возврата каретки
lf      equ     0ah             ; Код ASCII перевода строки
eom     equ     '$'             ; Признак конца сообщения

Header:                         ; Заголовок драйвера устройства
        dd      -1              ; Связь со следующим драйвером устройства
        dw      0c840h          ; Слово атрибутов устройства
        dw      Strat           ; Входная точка программы "стратегии"
        dw      Intr            ; Входная точка программы "прерывания"
        db      'SKELETON'      ; Имя логического устройства

RHPtr   dd      ?               ; Указатель на заголовок запроса,
                                ; передаваемый ядру MS-DOS программой
                                ; стратегии

Dispatch:                       ; Таблица-диспетчер командных кодов
                                ; программы прерывания
        dw      Init            ;  0 = инициализация драйвера
        dw      MediaChk        ;  1 = проверка носителя
        dw      BuildBPB        ;  2 = построение BPB
        dw      IoctlRD         ;  3 = IOCTL - чтение
        dw      Read            ;  4 = чтение
        dw      NdRead          ;  5 = неразрушающее чтение
```

Рис. 14.9. DRIVER.ASM. Функциональная схема, которая поможет вам в разработке собственного рабочего драйвера устройства.

```
        dw      InpStat         ;  6 = состояние ввода
        dw      InpFlush        ;  7 = очистка входных буферов
        dw      Write           ;  8 = запись
        dw      WriteVfy        ;  9 = запись с верификацией
        dw      OutStat         ; 10 = состояние вывода
        dw      OutFlush        ; 11 = очистка выходных буферов
        dw      IoctlWt         ; 12 = IOCTL - запись
        dw      DevOpen         ; 13 = устройство открыто  (MS-DOS 3.0+)
        dw      DevClose        ; 14 = устройство закрыто  (MS-DOS 3.0+)
        dw      RemMedia        ; 15 = сменный носитель    (MS-DOS 3.0+)
        dw      OutBusy         ; 16 = вывод до занятости  (MS-DOS 3.0+)
        dw      Error           ; 17 = не используется
        dw      Error           ; 18 = не используется
        dw      GenIOCTL        ; 19 = родовой IOCTL       (MS-DOS 3.2+)
        dw      Error           ; 20 = не используется
        dw      Error           ; 21 = не используется
        dw      Error           ; 22 = не используется
        dw      GetLogDev       ; 23 = получить логическое устройство
                                ; (MS-DOS 3.2+)
        dw      SetLogDev       ; 24 = установить логическое устройство
                                ; (MS-DOS 3.2+)

Strat   proc    far             ; Программа стратегии драйвера устройства
                                ; Вызывается ядром MS-DOS при
                                ; ES:BX = адрес заголовка запроса

                                ; Сохранить указатель заголовка запроса
        mov     word ptr cs:[RHPtr],bx
        mov     word ptr cs:[RHPtr+2],es

        ret                     ; Назад в ядро MS-DOS

Strat   endp


Intr    proc    far             ; Программа прерывания драйвера устройства
                                ; Вызывается ядром MS-DOS сразу
                                ; после вызова программы стратегии

        push    ax              ; Сохранить регистры общего назначения
        push    bx
        push    cx
        push    dx
        push    ds
        push    es
        push    di
        push    si
        push    bp

        push    cs              ; Сделаем локальные данные адресуемыми,
        pop     ds              ; установив DS = CS

        les     di,[RHPtr]      ; Пусть ES:DI = заголовок запроса

                                ; Получить BX = командный код
        mov     bl,es:[di+2]
```

Рис. 14.9.

```
        xor     bh,bh
        cmp     bx,MaxCmd       ; Удостоверимся, что он допустим
        jle     Intr1           ; Переход, функциональный код нормальный
        call    Error           ; Установить бит ошибки. Код "неизвестная
        jmp     Intr2           ; команда"

Intr1:  shl     bx,1            ; Сформируем индекс для поиска в таблице-
                                ; диспетчере и перейдем на программу
                                ; командного кода
        call    word ptr [bx+Dispatch]

        les     di,[RHPtr]      ; ES:DI = адрес заголовка запроса
Intr2:  or      ax,0100h        ; Включим бит "выполнено" в состояние и
        mov     es:[di+3],ax    ; сохраним состояние в заголовке запроса

        pop     bp              ; Восстановить регистры общего назначения
        pop     si
        pop     di
        pop     es
        pop     ds
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret                     ; Назад в ядро MS-DOS


; Программы командных кодов вызываются программой прерывания
; через таблицу-диспетчер. ES:DI указывает на заголовок
; драйвера. Каждая программа должна возвращать AX = 0,
; если функция выполнилась успешно, и AX = (8000h + код ошибки),
; если функция не выполнилась.

MediaChk proc   near            ; Функция 1 = проверка носителя
xor     ax,ax
        ret
Mediachk endp

BuildBPB proc   near            ; Функция 2 = построение BPB
        xor     ax,ax
        ret
BuildBPB endp

IoctlRd proc    near            ; Функция 3 = IOCTL - чтение
        xor     ax,ax
        ret
IoctlRd endp

Read    proc    near            ; Функция 4 = чтение(ввод)
        xor     ax,ax
        ret
Read    endp

Nread   proc    near            ; Функция 5 = неразрушающее чтение
        xor     ax,ax
        ret
Nread   endp
```

Рис. 14.9.

```
InpStat proc    near            ; Функция 6 = состояние ввода
        xor     ax,ax
        ret
InpStat endp

InpFlush                        Функция 7 = очистка входных буферов
        xor     ax,ax
        ret
InpFlush

Write   proc    near            ; Функция 8 = запись (вывод)
        xor     ax,ax
        ret
Write   endp

WriteVfy proc   near            ; Функция 9 = запись с верификацией
        xor     ax,ax
        ret
WriteVfy endp

OutStat proc    near            ; Функция 10 = состояние вывода
        xor     ax,ax
        ret
OutStat endp

OutFlush proc   near            ; Функция 11 = очистка выходных буферов
        xor     ax,ax
        ret
OutFlush endp

IoctlWt proc    near            ; Функция 12 = IOCTL - запись
        xor     ax,ax
        ret
IoctlWt endp

DevOpen proc    near            ; Функция 13 = устройство открыто (MS-DOS 3.0+)
        xor     ax,ax
        ret
DevOpen endp

DevClose proc   near            ; Функция 14 = устройство закрыто (MS-DOS 3.0+)
        xor     ax,ax
        ret
DevClose endp

RemMedia proc   near            ; Функция 15 = сменный носитель (MS-DOS 3.0+)
        xor     ax,ax
        ret
RemMedia endp

OutBusy proc    near            ; Функция 16 = вывод до занятости (MS-DOS 3.0+)
        xor     ax,ax
        ret
OutBusy endp

GenIOCTL proc   near            ;  Функция 19 = родовой IOCTL (MS-DOS 3.2+)
```

Рис. 14.9.

```
        xor     ax,ax
        ret
GenIOCTL endp

GetLogDev proc                  ; Функция 23 = получить логическое устройство
        xor     ax,ax           ; (MS-DOS 3.2+)
        ret
        GetLogDev endp

SetLogDev proc                  ; Функция 24 = установить логическое устройство
        xor     ax,ax           ; (MS-DOS 3.2+)
        ret
SetLogDev endp

Error   proc    near            ; Недопустимый командный код
                                ; в заголовке драйвера
        mov     ax,8003h        ; Бит ошибки + код "неизвестная команда"
        ret
Error   endp

Init    proc    near            ; Функция 0 = инициализация драйвера

        push    es              ; Сохранить адрес заголовка запроса
        push    di

        mov     ax,cs           ; Преобразовать адрес загрузки в ASCII
        mov     bx,offset Ident1
        call    hexasc

        mov     ah,9            ; Выдать сообщение-заголовок драйвера
        mov     dx,offset Ident
        int     21h

        pop     di              ; Восстановить адрес заголовка драйвера
        pop     es

                                ; Установить адрес свободной памяти
                                ; над драйвером (адрес разрыва)
        mov     word ptr es:[di+14],offset Init
        mov     word ptr es:[di+16],cs

        xor     ax,ax           ; Состояние возврата
        ret

Init    endp

hexasc  proc    near            ; Преобразование слова в шестнадцатеричное ASCII
                                ; При вызове AX = значение
                                ; DS:BX = адрес для строки
                                ; При возврате AX, BX уничтожаются

        push    cx              ; Сохранить регистры
        push    dx

        mov     dx,4            ; Инициализация счетчика символов

 hexasc1:
        mov     cx,4            ; Изолировать следующие четыре бита
```

Рис. 14.9.

```
        rol     ax,cl
        mov     cx,ax
        and     cx,0fh
        add     cx,'0'          ; Преобразовать в ASCII
        cmp     cx,'9'          ; Это 0 - 9?
        jbe     hexasc2         ; Да, переход
        add     cx,'A'-'9'-1    ; Добавить для получения букв A - F

hexasc2:
        mov     [bx],cl
        inc     bx              ; Инкремент указателя в строке

        dec     dx              ; Счетчик преобразованных символов
        jnz     hexasc1         ; Цикл, еще не четыре

        pop     dx              ; Восстановить регистры
        pop     cx
        ret                     ; Возврат в вызывающую программу

hexasc  endp

Ident   db      cr,lf,lf
        db      'Расширенный пример драйвера устройства MS-DOS'
        db      cr,lf
        db      'Заголовок драйвера устройства находится: '
Ident1  db      'XXXX:0000'
        db      cr,lf,eom

intr    endp

_TEXT   ends

        end
```

Рис. 14.9.

остерегайтесь каких-либо предположений в программных строках драйвера относительно его расположения в физической памяти. На рис. 14.9 приведен пример структуры драйвера, который будет полезен при чтении следующих нескольких страниц.

Заголовок драйвера устройства должен располагаться в начале файла (относительный адрес 0000H). Оба слова в поле связи заголовка должны содержать —1. В слове атрибутов следует правильно указать тип устройства и другие параметры. Смещения к программам стратегии и прерываний должны вычисляться относительно той же базы сегмента, что и сам драйвер. Если драйвер предназначен для символьно-ориентированного устройства, поле name должно содержать логическое имя устройства, которое образуется так же, как и имя файла, и состоит из 8 символов, дополненных пробелами (без двоеточия). Остерегайтесь случайно повторить имя существующего символьно-ориентированного устройства, если только вы не имеете в виду заменить резидентный драйвер собственным.

MS-DOS вызывает программы стратегии и прерываний, относящиеся к конкретному устройству, с помощью межсегментного вызова (CALL FAR). Первый раз это делается, когда загружается и устанавливается драйвер, а затем каждый раз при запросе прикладной программы на ввод-вывод к данному устройству. MS-DOS использует регистры ES:BX для передачи программе strat двухсловного указателя заголовка запроса; драйвер должен сохранить этот адрес, чтобы его можно было использовать при последующем вызове программы intr.

Программы командных кодов с функциональными кодами от 0 до 12 (0CH) должны присутствовать в каждом устанавливаемом драйвере устройства независимо от типа устройства. Функции с кодами 13 (0DH) и выше необязательны и используются в драйверах, работающих в среде MS-DOS начиная с версии 3.0, одним из следующих способов:

- Не используйте их и оставьте сброшенными соответствующие им биты в заголовке устройства. В этом случае драйвер будет работать и в версии 2, и в версии 3, однако расширенные возможности версии 3 реализованы не будут.
- Используйте их и предусмотрите анализ версии MS-DOS в процессе инициализации, установив соответствующим образом биты 6 и 11 заголовка устройства. Пишите все программы командных кодов таким образом, чтобы они анализировали этот бит и настраивались на работающую версию MS-DOS. Такой драйвер потребует больше работы при отладке и тестировании, но зато полностью реализует возможности версий 2 и 3.
- Используйте их в предположении, что все средства версии 3 доступны. В этом случае в среде версии 2 драйвер, возможно, будет работать неправильно.
- Не забудьте, что драйверы устройств должны сохранять целостность MS-DOS. Драйвер должен сохранять и восстанавливать все регистры, включая флаги (особенно флаг направления и биты разрешения прерываний), и в случае интенсивного использования стека драйвера должен переключаться на внутренний стек достаточной глубины (стек MS-DOS при вызове драйвера имеет свободного места лишь от 40 до 50 байт).

При установке нового драйвера CON не забудьте установить биты стандартного ввода и стандартного вывода слова атрибутов устройства в заголовке драйвера.

Как уже упоминалось, один файл может содержать несколько драйверов. В этом случае поле связи заголовка каждого драйвера должно указывать на относительный адрес следующего драйвера в сегменте при одной и той же базе сегмента, а поле связи последнего драйвера должно содержать −1, −1. Программы инициализации всех драйверов в файле должны возвращать один и тот же адрес разрыва.

Компоновка

Для преобразования файла .OBJ (результата ассемблирования) в перемещаемый модуль .EXE используйте стандартный компоновщик MS-DOS. Затем с помощью утилиты EXE2BIN (см. гл. 4) преобразуйте файл .EXE в образ памяти. Расширение результирующего файла с драйвером может быть любым, но обычно в системах MS-DOS используются расширения .BIN и .SYS, и мы рекомендуем не нарушать эти соглашения.

Установка

После ассемблирования, компоновки и преобразования драйвера в файл .BIN или .SYS скопируйте его в корневой каталог загружаемого диска. Если драйвер предназначен для символьно-ориентированного устройства, не используйте для файла то же имя, которое дано логическому устройству в заголовке драйвера, так как в этом случае после загрузки драйвера вы не сможете удалять, копировать и переименовывать файл.

С помощью вашего любимого текстового редактора добавьте в файл CONFIG.SYS на загружаемом диске строку

DEVICE = [D:] [путь] имя_файла.расширение

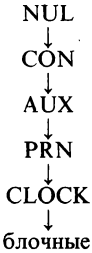
В этой строке D: – необязательный идентификатор диска, а имя файла.расширение –

имя файла, содержащего ваш новый драйвер устройства. Если вы предпочитаете держать файл с драйвером в каталоге, отличном от корневого, можно добавить описание пути.

При выполнении процедуры загрузки MS-DOS модуль SYSINIT (являющийся частью IO.SYS) читает и выполняет файл CONFIG.SYS. Он загружает драйвер в память и анализирует заголовок устройства. Если драйвер предназначен для символьно-ориентированного устройства, SYSINIT включает его в цепочку устройств перед другими символьно-ориентированными устройствами; если драйвер предназначен для блочно-ориентированных устройств, SYSINIT располагает его за цепочкой связанных блочно-ориентированных устройств и за резидентными блочно-ориентированными устройствами (рис. 14.10, 14.11 и 14.12). SYSINIT завершает организацию связей, модифицируя поле связи в заголовке устройства так, чтобы оно указывало на сегмент и относительный адрес следующего драйвера в цепочке. Поле связи последнего драйвера в цепочке содержит -1, -1.

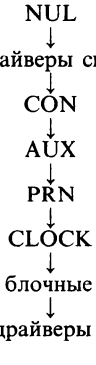
Далее SYSINIT вызывает программу strat с заголовком запроса, содержащим командный код 0, после чего вызывает программу intr. Драйвер выполняет программу инициализации и возвращает адрес разрыва, информируя MS-DOS об объеме памяти, который необходимо зарезервировать для этого драйвера. После этого MS-DOS может перейти к следующей записи в файле CONFIG.SYS.

Вы не можете заменить встроенный драйвер блочно-ориентированного устройства – можно только добавлять дополнительные блочно-ориентированные устройства. Однако стандартный системный драйвер для символьно-ориентированного устройства



Любые другие резидентные блочные или символьные устройства

Рис. 14.10. Цепочка драйверов устройств MS-DOS перед загрузкой устанавливаемых драйверов устройств.



Устанавливаемые драйверы символьных устройств

Любые другие резидентные блочные или символьные устройства

Устанавливаемые драйверы блочных устройств

Рис. 14.11. Цепочка драйверов устройств MS-DOS после загрузки устанавливаемых драйверов устройств.

Адрес	Атрибут	Программа стратегии	Программа прерывания	Тип	Число устройств	Имя
00E3:0111	8004	0FD5	0FE0	C		NUL
0070:0148	8013	008E	0099	C		CON
0070:01DD	8000	008E	009F	C		AUX
0070:028E	8000	008E	00AE	C		PRN
0070:0300	8008	008E	00C3	C		CLOCK
0070:03CC	0000	008E	00C9	B	02	
0070:01EF	8000	008E	009F	C		COM1
0070:02A0	8000	008E	00AE	C		LPT1
0070:06F0	8000	008E	00B4	C		LPT2
0070:0702	8000	008E	00BA	C		LPT3
0070:0714	8000	008E	00A5	C		COM2
Конец цепочки устройств						

Рис. 14.12. Пример цепочки устройств для MS-DOS версии 2.1 и «простенькой» машины IBM PC без жестких дисков или пользовательских драйверов устройств. (С – символьно-ориентированное устройство, В – блочно-ориентированное устройство.)

(например, CON) можно заменить на установленный драйвер, если дать ему то же самое имя логического устройства, указываемое в заголовке устройства. При обработке запроса на символьный ввод-вывод MS-DOS всегда сначала просматривает список установленных драйверов и лишь затем – список стандартных устройств и руководствуется первым совпадением.

ОТЛАДКА ДРАЙВЕРА УСТРОЙСТВА

Важнейшие требования, которые надо иметь в виду при тестировании новых драйверов устройств, – обеспечение получения необходимых резервных копий и возможность вернуть систему в исходное состояние. Не модифицируйте файл CONFIG.SYS и не устанавливайте новый драйвер на вашем жестком диске до его полной проверки! Будьте предусмотрительны – создайте загружаемую дискету и поместите на нее модифицированный файл CONFIG.SYS и новый драйвер для отладки. Когда все будет работать должным образом, скопируйте конечный продукт на постоянное место хранения.

Простейший способ тестирования нового драйвера устройства заключается в написании простой интерфейсной программы на языке ассемблера, которая создает модельный пакет запроса, а затем выполняет дальние вызовы (FAR CALL) точек входа strat и intr точно так же, как это делала бы MS-DOS. После этого вы можете скомпоновать драйвер вместе со своим интерфейсом в файл .COM или .EXE, который можно запустить под управлением CodeView или другого отладчика. Такая организация тестирования облегчает индивидуальную трассировку каждой программы командного кода, наблюдение результатов ввода-вывода и проверку кодов состояния, возвращаемых в заголовке запроса.

Трассировка установленного драйвера, когда он обычным образом скомпонован с системой MS-DOS, – более трудная задача. Следует весьма тщательно расставить точки останова, чтобы в каждом отладочном прогоне получить максимум информации. Из-за того что существующие версии MS-DOS внутренне поддерживают только один заголовок запроса, заголовок запроса, используемый трассируемым драйвером, будет затерт, как только отладчик сделает запрос на вывод информации на экран дисплея. Весьма полезно добавить к вашей подпрограмме инициализации программу, выводящую на консоль при начальной загрузке MS-DOS адрес загрузки драйвера; этот

адрес можно затем использовать для анализа заголовка драйвера устройства и установки точек останова в теле драйвера.

Отладка драйвера устройства может оказаться весьма неприятным делом, когда включает в себя обслуживание прерываний, особенно если устройство использует тот же уровень приоритета запросов прерываний (уровень IRQ), что и остальная периферия системы. Аккуратное, консервативное программирование помогает избежать неожиданных и невоспроизводимых взаимодействий с драйверами других устройств и обработчиками прерываний. Если возможно, отладьте основную логику работы драйвера с использованием ввода-вывода в режиме опроса, а не в режиме прерываний, а обслуживание прерываний вводите, только когда убедитесь в правильной работе остальных элементов драйвера.

Можно перечислить следующие типичные ошибки драйверов устройств, которые могут привести к аварии системы или непредсказуемому ее поведению:

- Ошибочная установка адреса связи последнего драйвера в файле, не равного −1.
- Переполнение стека MS-DOS программой инициализации драйвера, в результате чего повреждается образ MS-DOS в памяти (такая ошибка приводит к непредсказуемым действиям в процессе начальной загрузки; рецепт – использование локального стека).
- Неправильный адрес разрыва, переданный программой инициализации (может привести к аварии системы, если следующий загружаемый драйвер затирает важные поля предыдущего).
- Неправильные блоки параметров BIOS (BPB), созданные программой построения BPB, или неправильный массив указателей BPB, созданный программой инициализации (это может привести к разнообразным неприятностям, начиная от ошибок выхода за пределы памяти до аварии системы в процессе начальной загрузки).
- Неправильное сообщение о числе байтов или секторов, успешно переданных к моменту ошибки ввода-вывода (может привести к аварии системы, если в ответ на запрос системы Abort, Retry, Ignore вы отвечаете R (повторить)).

Хотя интерфейс между ядром DOS и драйвером устройства весьма прост, он, с другой стороны, должен неукоснительно выдерживаться. Программы командных кодов должны выполняться в точности так, как для них определено, иначе поведение системы станет непредсказуемым. Даже малейшее отклонение от нормы в действиях программы командных кодов может привести к неожиданно значительным глобальным эффектам.

ГЛАВА 15. ФИЛЬТРЫ

Фильтр – это прежде всего программа, взаимодействующая с потоком символов. Источником и приемником потока символов могут быть файлы, другая программа или почти любое символьно-ориентированное устройство. Преобразования потока символов, выполняемые фильтром, могут быть как очень простыми (например, замена символа другим), так и весьма сложными (генерация сплайнов по набору координат точек).

Стандартный пакет MS-DOS включает три простых фильтра: SORT, сортирующий строки текста по алфавиту; FIND, выполняющий поиск указанной строки в потоке текста; MORE, организующий постраничный вывод на экран терминала.

СИСТЕМНАЯ ПОДДЕРЖКА ФИЛЬТРОВ

Работа программ фильтрации опирается на два средства MS-DOS, впервые введенные в версии 2.0, – стандартные устройства и перенаправление ввода-вывода.

Стандартные устройства представлены пятью дескрипторами, первоначально устанавливаемыми программой COMMAND.COM. Каждый процесс наследует эти дескрипторы от своего непосредственного родителя. Таким образом, когда процесс принимает на себя управление системой, стандартные дескрипторы устройств уже открыты, и процесс может их использовать для выполнения чтения и записи без каких-либо подготовительных операций через прерывание Int 21H с функциями 3FH и 40H. Действующие по умолчанию назначения стандартных дескрипторов устройств приведены ниже:

Указатель	**Имя**	**Устройство по умолчанию**
0	stdin (стандартный ввод)	CON
1	stdout (стандартный вывод)	CON
2	stderr (стандартная ошибка)	CON
3	stdaux (стандартный дополнительный порт)	AUX
4	stdprn (стандартный принтер)	PRN

Устройство CON по умолчанию назначается клавиатуре и экрану системного дисплея. AUX и PRN по умолчанию связаны с COM1 (первый физический последовательный порт) и LPT1 (первый параллельный порт принтера) соответственно. С помощью команды MODE можно перенаправить LPT1 на один из последовательных портов; команда MODE переназначает также и PRN.

Запуская программу указанием ее имени в ответ на запрос COMMAND.COM, вы можете перенаправить стандартный ввод, стандартный вывод или и то и другое с действующего по умолчанию устройства (CON) другому файлу, символьно-ориентированному устройству или процессу. Это делается включением в командную строку одного из специальных символов <, >, > и | в соответствии с приведенными ниже форматами:

Символ	**Действие**
<файл	Стандартный ввод осуществляется не с клавиатуры, а из указанного файла *файл*
<устройство	Стандартный ввод осуществляется не с клавиатуры, а с устройства *устройство*
>файл	Стандартный вывод направляется не на дисплей, а в указанный файл *файл*
≫файл	Стандартный вывод направляется не на дисплей, а в указанный файл *файл*, добавляясь к его содержимому
>устройство	Стандартный вывод направляется не на дисплей, а на указанное устройство *устройство*
p1|p2	Стандартный вывод программы p1 используется как стандартный ввод программы p2 (говорят, что образуется конвейер программ p1 и p2)

Например, команда

C > SORT < MYFILE.TXT > PRN<Enter>

заставляет фильтр SORT прочитать в качестве входного текста файл MYFILE.TXT,

осуществить сортировку его строк в алфавитном порядке и вывести результирующий текст на символьно-ориентированное устройство PRN (логическое имя для системного устройства печати).

Перенаправление по запросам <, >, >> и | осуществляется на уровне COMMAND.COM и невидимо для программы, на которую направлено это действие. Любой другой процесс может добиться того же результата, перенаправляя стандартный ввод и стандартный вывод с помощью прерывания Int 21H с функцией 46H перед вызовом функции EXEC (Int 21H с функцией 4BH) для запуска дочернего процесса.

Заметьте, что в случаях, когда программа обходит MS-DOS при выполнении ввода и вывода (либо вызывая функции ПЗУ BIOS, либо непосредственно управляя контроллером дисплея), команды перенаправления, введенные в командную строку вызова программы, не дают ожидаемого эффекта.

КАК РАБОТАЕТ ФИЛЬТР

По установленному соглашению программа фильтра читает входной текст из устройства стандартного ввода и записывает результат своей работы на устройство стандартного вывода. Обнаружив окончание входного потока, фильтр попросту завершает свою работу. В результате фильтры оказываются одновременно и гибкими, и простыми.

Программы фильтров являются гибкими, потому что они не знают и не желают знать об источнике обрабатываемых ими данных или о приемнике результатов их работы. В результате любое символьно-ориентированное устройство, имеющее в системе логическое имя (CON, AUX, COM1, COM2, PRN, LPT1, LPT2, LPT3 и т.д.), любой файл на любом блочно-ориентированном устройстве (локальном или сетевом), известном системе, или любая программа могут служить источником входных данных для фильтра или воспринимать результаты его работы. При необходимости можно сцепить несколько функционально простых фильтров с помощью символов конвейеризации и выполнять в результате весьма сложные операции.

Будучи гибкими, фильтры, тем не менее, просты, так как они ожидают от породивших их процессов предоставления перенаправленных должным образом указателей стандартного ввода и стандартного вывода. Родительский процесс должен открыть или создать все необходимые файлы, проверить правильность имен логических символьно-ориентированных устройств, а также загрузить и выполнить предыдущий и последующий процессы в конвейере. Сам фильтр выполняет лишь преобразование данных.

ПОСТРОЕНИЕ ФИЛЬТРА

Создание нового фильтра для MS-DOS является довольно простым делом. В простейшей форме фильтр с помощью функций ввода-вывода типа дескрипторов (Int 21H с функцией 3FH для чтения и Int 21H с функцией 40H для записи) получает символы или строки стандартного ввода и направляет их в стандартный вывод, выполняя любые требуемые изменения входного потока посимвольно или построчно.

На рис. 15.1 и 15.2 приведены модельные варианты символьно-ориентированных фильтров на языках ассемблера и Си. В этих примерах программа преобразования символов translate, вызываемая для каждого символа, передаваемого из стандартного ввода в стандартный вывод, не делает ничего. В результате оба фильтра функционируют как очень медленная команда COPY. Вы можете без труда преобразовать эти примитивные фильтры в полезные программы, подставив собственную программу преобразования translate.

```
        name    proto
        page    55,132
        title   PROTO.ASM--prototype filter
                        ;Модельный фильтр
;
; PROTO.ASM: Модельный символьно-ориентированный фильтр
;
; Авторские права 1988 Рей Данкан
;

stdin   equ     0               ; Дескриптор стандартного ввода
stdout  equ     1               ; Дескриптор стандартного  вывода
stderr  equ     2               ; Дескриптор стандартной ошибки

cr      equ     0dh             ; Код ASCII возврата каретки
lf      equ     0ah             ; Код ASCII перевода строки

_TEXT   segment word public 'CODE'

        assume  cs:_TEXT,ds:_DATA,ss:STACK

main    proc    far             ; Точка входа из MS-DOS

        mov     ax,_DATA        ; Установить DS = наш сегмент данных
        mov     ds,ax

main1:                          ; Прочитаем символ из stdin...
        mov     dx,offset char  ; DS:DX = адрес буфера
        mov     cx,1            ; CX  = длина читаемого
        mov     bx,stdin        ; BX = дескриптор стандартного ввода
        mov     ah,3fh          ; Функция 3fh = чтение
        int     21h             ; Переход в MS-DOS
        jc      main3           ; Если ошибка,  завершить

        cmp     ax,1            ; Какой-либо символ прочитан?
        jne     main2           ; Если конец файла, завершить

        call    translate       ; Транслировать символ

                                ; Запишем символ в stdout
        mov     dx,offset char  ; DS:DX = адрес буфера
        mov     cx,1            ; CX = длина записываемого
        mov     bx,stdout       ; BX = дескриптор стандартного вывода
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS
        jc      main3           ; Если ошибка - завершить
        cmp     ax,1            ; Символ был записан?
        jne     main3           ; Если диск полон, завершить

        jmp     main1           ; Получить следующий символ

main2:                          ; Достигнут конец файла
        mov     ax,4c00h        ; Функция 4ch = завершить
                                ; Код возврата = 0
        int     21h             ; Переход в MS-DOS

main3:                          ; Ошибка или диск полон
```

Рис. 15.1. PROTO.ASM – исходный текст модельного символьно-ориентированного фильтра на языке ассемблера.

```
        mov     ax,4c01h        ; Функция 4ch = завершить
                                ; Код возврата = 1
        int     21h             ; Переход в MS-DOS

main    endp

;
; Выполните требуемую трансляцию символа, полученного из
; стандартного ввода и записанного в переменную "char".
; В этом примере символ просто остается каким был.
;

translate proc near

        ret                     ; Ничего не делает

translate endp

_TEXT   ends

_DATA   segment word public 'DATA'

char    db      0               ; Место для хранения введенного символа

_DATA   ends

STACK   segment para stack 'STACK'

        dw      64 dup (?)

STACK   ends

        end     main            ; Определяет точку входа в программу
```

Рис. 15.1.

```
/*
        PROTO.C: Модельный символьно-ориентированный фильтр

        Авторские права 1988 Рей Данкан
*/

#include <stdio.h>

main(int argc, char *argv[])
{
        char ch;

        while((ch=getchar()) !=EOF) /* Прочитать символ            */
        {
             ch = translate(ch);    /* Транслировать, если нужно   */

             putchar(ch);           /* Записать символ             */
        }
        exit(0);                    /* Закончить при конце файла   */

}
```

Рис. 15.2. PROTO.C – исходный текст символьно-ориентированного фильтра на языке Си.

```
/*
    Выполните требуемую трансляцию символа, полученного из
    входного файла. В этом примере символ просто остается
    каким был.
*/
int translate(char ch)
{
        return (ch)
}
```

Рис. 15.2.

Если вы испытаете приведенные программы, то увидите, что модельный фильтр, написанный на языке Си, работает значительно быстрее своего эквивалента на языке ассемблера. Это связано с тем, что исполняющая библиотека (runtime library) Си выполняет скрытое блокирование (в смысле объединения в блок) и разблокирование входного и выходного потоков, в то время как фильтр на языке ассемблера делает в точности то, что в нем написано: дважды вызывает MS-DOS для любого обрабатываемого символа. Вы можете без труда реализовать в фильтре на языке ассемблера ожидаемые высокие скоростные характеристики, изменив его так, чтобы вместо отдельных символов читались и записывались целые строки.

ФИЛЬТР CLEAN

В качестве более реального примера фильтра MS-DOS рассмотрим простой, но весьма полезный фильтр под названием CLEAN. На рис. 15.3 и 15.4 приведены

```
        name    clean
        page    55,132
        title   CLEAN--Text-file filter
                        ; Фильтр текстового файла
;
; CLEAN.ASM       Фильтр для преобразования файлов документов
;                 в обычные текстовые файлы
; Авторские права 1988 Рей Данкан
;
; Компоновка:     C>MASM CLEAN;
;                 C>LINK CLEAN;
;
; Использование: C>CLEAN   <входной_файл >выходной_файл
;
; Все проходящие через фильтр текстовые символы лишаются
; старшего бита. Коды перевода страницы, возврата каретки
; и перевода строки проходят через фильтр. Коды табуляции
; преобразуются в требуемое число пробелов. Все остальные
; управляющие коды отбрасываются.
;

tab     equ     09h             ; Код ASCII табуляции
lf      equ     0ah             ; Код ASCII перевода строки
ff      equ     0ch             ; Код ASCII перевода страницы
cr      equ     0dh             ; Код ASCII возврата каретки
blank   equ     020h            ; Код ASCII пробела
eof     equ     01ah            ; Конец файла (Ctrl-Z)
```

Рис. 15.3. CLEAN.ASM – исходный текст программы фильтра CLEAN на языке MASM.

```
tabsiz  equ     8               ; Интервал табуляции

bufsiz  equ     128             ; Размер входного и
                                ; выходного буферов

stdin   equ     0000            ; Дескриптор стандартного ввода
stdout  equ     0001            ; Дескриптор стандартного вывода
stderr  equ     0002            ; Дескриптор стандартной ошибки

_TEXT   segment word public 'CODE'

        assume  cs:_TEXT,ds:_DATA,es:_DATA,ss:STACK

clean   proc    far             ; Точка входа из MS-DOS

        push    ds              ; Сохранить DS:0000 для завершающего
        xor     ax,ax           ; возврата в MS-DOS на случай невозможности
        push    ax              ; использовать функцию 4ch
        mov     ax,_DATA        ; Сделаем сегмент данных адресуемым
        mov     ds,ax
        mov     es,ax

        mov     ah,30h          ; Проверить версию MS-DOS
        int     21h             ; Переход в MS-DOS
        cmp     al,2            ; MS-DOS версии 2.0 и выше?
        jae     clean1          ; Переход, если версия годится

                                ; MS-DOS 1, выдать сообщение об ошибке
                                ; и на выход...
        mov     dx,offset msg1  ; DS:DX = адрес сообщения
        mov     ah,9            ; Функция 9 = вывести строку на экран
        int     21h             ; Переход в MS-DOS
        ret                     ; Выход обычным образом

clean1: call    init            ; Инициализация входного буфера

clean2: call    getc            ; Получить входной символ
        jc      clean9          ; Выход, если конец входного потока

        and     al,07fh         ; Убрать старший бит

cmp     al,blank        ; Это управляющий символ?
jae     clean4          ; Нет, записать его

cmp     al,eof          ; Это конец файла?
je      clean8          ; Да, записать EOF и на выход

cmp     al,tab          ; Это символ табуляции?
je      clean6          ; Да, преобразовать его в пробелы

cmp     al,cr           ; Это возврат каретки?
je      clean3          ; Да, на обработку его

cmp     al,lf           ; Это перевод строки?
je      clear3          ; Да, на обработку его

cmp     al,ff           ; Это перевод страницы?
jne     clean2          ; Нет, отбросить его
```

Рис. 15.3.

```
clean3: mov     column,0        ; Если CR, LF или FF,
        jmp     clean5          ; вернуться к нулевому столбцу

clean4: inc     column          ; Если не управляющий символ,
                                ; инкремент счетчика столбца

clean5: call    putc            ; Записать символ в стандартный вывод
        jnc     clean2          ; Если диск не полон,
                                ; получить следующий символ

                                ; Запись не прошла...
        mov     dx,offset msg2  ; DS:DX = сообщение об ошибке
        mov     cx,msg2_len     ; CX = длина сообщения
        mov     bx,stderr       ; BX = дескриптор стандартной ошибки
        mov     ah,40h          ; Функция 40h = запись
        int     21h             ; Переход в MS-DOS

        mov     ax,4c01h        ; Функция 4ch = завершить
                                ; Код возврата = 1
        int     21h             ; Переход в MS-DOS

clean6: mov     ax,column       ; Обнаружен код табуляции
        cwd                     ; tabsiz - (column MOD tabsiz)
        mov     cx,tabsiz       ; это число пробелов, требуемое
        idiv    cx              ; для перехода на следующую позицию табуляции
        sub     cx,dx

        add     column,cx       ; Также обновим счетчик столбцов

clean7: push    cx              ; Сохранить счетчик пробелов

        mov     al,blank        ; Записать пробел ASCII
        call    putc



        pop     cx              ; Восстановить счетчик пробелов
        loop    clean7          ; Повторять до позиции табуляции

        jmp     clean2          ; Получить следующий символ

clean8: call    putc            ; Записать метку конца файла EOF

clean9: call    flush           ; Записать последний выходной буфер
        mov     ax,4c00h        ; Функция 4ch = завершить
                                ; Код возврата = 0
        int     21h             ; Переход в MS-DOS

clean   endp

getc    proc    near            ; Получает символ из стандартного ввода
                                ; Возвращает перенос = 1 в случае завершения
                                ; ввода, в противном случае
                                ; AL = символ, перенос = 0

        mov     bx,iptr         ; Получить указатель входного буфера
        cmp     bx,ilen         ; Достигнут конец буфера?
        jne     getc1           ; Нет еще, переход
```

Рис. 15.3.

```
                                ; Требуются дополнительные данные...
        mov     bx,stdin        ; BX = дескриптор стандартного ввода
        mov     cx,bufsiz       ; CX = длина читаемого
        mov     dx,offset ibuff ; DS:DX = адрес буфера
        mov     ah,3fh          ; Функция 3fh = прочитать
        int     21h             ; Переход в MS-DOS
        jc      getc2           ; Переход, если чтение не выполнилось

        or      ax,ax           ; Что-нибудь было прочитано?
        jz      getc2           ; Переход, если конец ввода

        mov     ilen,ax         ; Сохранить длину данных
        xor     bx,bx           ; Сбросить указатель буфера

getc1:  mov     al,[ibuff+bx]   ; Получить символ из буфера
        inc     bx              ; Инкремент указателя буфера

        mov     iptr,bx         ; Сохранить обновленный указатель
        clc                     ; Возвращаемый символ в AL,
        ret                     ; а перенос = 0 (сброшен)

getc2:  stc                     ; Конец входного потока
        ret                     ; При возврате перенос = 1 (установлен)

getc    endp


putc    proc    near            ; Посылает символ в стандартный вывод
                                ; Возвращает перенос = 1, если ошибка
                                ; В противном случае перенос = 0


        mov     bx,optr         ; Записать символ
        mov     [obuff+bx],al   ; в выходной буфер

        inc     bx              ; Инкремент указателя буфера
        cmp     bx,bufsiz       ; Буфер полон?
        jne     putc1           ; Нет, переход


        mov     bx,stdout       ; BX = дескриптор стандартного ввода
        mov     cx,bufsiz       ; CX = длина для записи
        mov     dx,offset obuff ; DS:DX = адрес буфера
        mov     ah,40h          ; Функция 40h = записать
        int     21h             ; Переход в MS-DOS
        jc      putc2           ; Переход, если запись не выполнилась

        cmp     ax,cx           ; Записалось все?
        jne     putc2           ; Переход, если диск полон

        xor     bx,bx           ; Сбросить указатель буфера

putc1:  mov     optr,bx         ; Сохранить указатель буфера
        clc                     ; Запись прошла успешно,
        ret                     ; вернуть перенос = 0 (сброшен)

putc2:  stc                     ; Запись не выполнилась  или диск полон
        ret                     ; Вернуть перенос = 1 (установлен)
```

Рис. 15.3.

```
putc    endp


init    proc    near            ; Инициализация входного буфера

        mov     bx,stdin        ; BX = дескриптор стандартного ввода
        mov     cx,bufsiz       ; CX = длина для чтения
        mov     dx,offset ibuff ; DS:DX = адрес буфера
        mov     ah,3fh          ; Функция 3fh = прочитать
        int     21h             ; Переход в MS-DOS
        jc      init1           ; Переход, если чтение не выполнилось
        mov     ilen,ax         ; Сохранить число реально прочитанных байтов
init1:  ret

init    endp


flush   proc    near            ; Очистка выходного буфера

        mov     cx,optr         ; CX = число байтов для записи
        jcxz    flush1          ; На выход, если буфер пуст
        mov     dx,offset obuff ; DS:DX = адрес буфера
        mov     bx,stdout       ; BX = дескриптор стандартного вывода
        mov     ah,40h          ; Функция 40h = записать
        int     21h             ; Переход в MS-DOS
flush1: ret

flush   endp

-TEXT   ends

-DATA   segment word public 'DATA'

ibuff   db      bufsiz dup (0)  ; Входной буфер
obuff   db      bufsiz dup (0)  ; Выходной буфер

iptr    dw      0               ; Указатель ibuff
ilen    dw      0               ; Байтов в ibuff
optr    dw      0               ; Указатель obuff

column  dw      0               ; Счетчик текущего столбца

msg1    db      cr,lf
        db      'clean: требуется MS-DOS версии 2 или выше.'
        db      cr,lf,'$'

msg2    db      cr,lf
        db      'clean: диск заполнен.'
        db      cr,lf
msg2_len equ    $-msg2

_DATA   ends

STACK   segment para stack 'STACK'

        dw      64 dup (?)

STACK   ends

        end     clean
```

Рис. 15.3.

```
/*
        CLEAN.C         Фильтр для преобразования файлов документов
                        в обычные текстовые файлы.

        Авторские права 1988 Рей Данкан

        Компиляция:     C>CL CLEAN.C

        Использование: C>CLEAN   <входной_файл >выходной_файл

        Все проходящие через фильтр текстовые символы лишаются
        старшего бита. Коды перевода страницы, возврата каретки
        и перевода строки проходят через фильтр. Коды табуляции
        преобразуются в требуемое число пробелов. Все остальные
        управляющие коды отбрасываются.
*/

#include <stdio.h>

#define TAB_WIDTH   8           /* Интервал табуляции              */
#define TAB     '\x09'          /* Код ASCII табуляции             */
#define LF      '\x0A'          /* Код ASCII перевода строки       */
#define FF      '\x0C'          /* Код ASCII перевода страницы     */
#define CR      '\x0D'          /* Код ASCII возврата каретки      */
#define BLANK   '\x20'          /* Код ASCII пробела               */
#define EOFMK   '\x1A'          /* Конец файла (Ctrl-Z)            */

main(int argc, char *argv[])
{
    char c;                     /* Символ из стандартного ввода    */
    int col = 0;                /* Счетчик столбцов                */

    while((c = getchar())!=EOF) /* Прочитать входной символ        */
    {
        c &= 0x07F;             /* Убрать старший бит              */

        switch(c)               /* Декодировать символ             */
        {
            case LF:            /* Если перевод строки             */
            case CR:            /* или возврат каретки,            */
                col=0;          /* сбросить счетчик столбцов       */

            case FF:            /* Если перевод страницы, возврат  */
                wchar(c);       /* каретки или перевод строки,     */
                break;          /* пропустить символ через фильтр  */

            case TAB:           /* Если табуляция, преобразовать
                                   в пробелы                       */
                do wchar(BLANK);
                while((++col % TAB_WIDTH) != 0);
                break;

            default:            /* Отбросить другие управляющие    */
                if(c >= BLANK)  /* символы, пропустить текстовые   */
                {               /* символы через фильтр            */
                    wchar(c);
                    col++;      /* Инкремент счетчика столбцов     */
```

Рис. 15.4. CLEAN.C – исходный текст программы фильтра CLEAN на языке Си.

```
                }
                break;
            }
        }
        wchar(EOFMK);                /* Записать метку конца файла      */
        exit(0);
}

/*
    Вывести символ на устройство стандартного вывода. Если фиксируется
    ошибка вывода, вывести на экран аварийное сообщение и завершить задачу.
/*

wchar(char c)
{
    if((putchar(c) == EOF) && (c != EOFMK))
    {
        fputs("clean: диск полон",stderr);
        exit(1);
    }
}
```

Рис. 15.4.

исходные тексты этого фильтра на языках ассемблера и Си. Фильтр CLEAN в процессе обработки текстового потока убирает старшие биты всех символов, расширяет символы табуляции в требуемое число пробелов и удаляет все управляющие коды, за исключением кодов возврата каретки, перевода строки и перевода страницы. Соответственно фильтр CLEAN («очистка») может преобразовать почти любой документ, полученный с помощью текстового процессора, в текстовый файл, состоящий из одних лишь кодов ASCII.

Используя фильтр CLEAN, вы должны описать файлы источника и приемника вместе с параметрами перенаправления в командной строке; в противном случае CLEAN будет вводить текст с клавиатуры и выводить его на экран. Например, для фильтрации документа в файле MYFILE.DOC и вывода результата в файл MYFILE.TXT следует ввести следующую команду:

C > CLEAN <MYFILE.DOC >MYFILE.TXT Enter

(Обратите внимание на то, что исходный файл MYFILE.DOC остается неизменным.)

Одно из полезных применений этого фильтра заключается в спасении исходных текстов на языке ассемблера. Если вы случайно отредактируете такой текст в режиме документа, результирующий файл при его трансляции ассемблером даст большое количество сообщений о малоприятных и, в сущности, отсутствующих ошибках. Программа CLEAN позволит вам вернуть файлу с исходным текстом форму, воспринимаемую ассемблером, без затрат времени на обратное редактирование текста.

Другое удобное приложение программы CLEAN – вывод документа, обработанного текстовым процессором, на принтер в исходном виде с помощью команды вида

C > CLEAN <MYFILE.DOC >PRN Enter

Сравнение приведенных вариантов фильтра на языках ассемблера и Си дает любопытные результаты. Вариант на Си содержит 79 строк и после трансляции образует файл .EXE объемом 5889 байт, в то время как вариант на языке ассемблера

содержит 265 строк и преобразуется в файл .EXE объемом 1107 байт. Преимущества языка ассемблера в отношении как размера программы, так и скорости ее выполнения при реализации такого рода инструментальных средств вполне очевидны, даже при сравнении с таким великолепным компилятором с языка Си, как Microsoft C Optimizing Compiler. Однако при оценке выигрыша в производительности следует рассмотреть время и затраты, неизбежные при программировании на языке ассемблера, особенно когда речь идет о программах, которые будут использоваться не слишком часто.

ГЛАВА 16. СОВМЕСТИМОСТЬ И ПЕРЕНОСИМОСТЬ

В начале книги мы привели исторический обзор MS-DOS, из которого следует, что новые версии системы выпускаются почти каждый год, будучи мало связанными с разработкой новых моделей персональных компьютеров. Затем было сказано об основных направлениях прикладного программирования для MS-DOS: интерфейс пользователя; массовая память; управление памятью; управление «дочерними» процессами; специальные виды программ, такие, как фильтры, обработчики прерываний и драйверы устройств.

Теперь наступило время замкнуть круг и рассмотреть два общих аспекта программирования в среде MS-DOS: совместимость и переносимость. Для того чтобы ваши программы сохраняли свою ценность в условиях непрерывно изменяющейся программной и аппаратной среды, вам следует разрабатывать их таким образом, чтобы они надежно выполнялись на любой разумной машинной конфигурации и использовали доступные системные ресурсы; кроме того, программы должны легко допускать усовершенствование, чтобы их можно было приспособить к новым версиям MS-DOS, новым машинам, а также и к совершенно новой операционной среде, такой, как MS OS/2.

СТЕПЕНИ СОВМЕСТИМОСТИ

Если мы посмотрим, как существующее прикладное программное обеспечение MS-DOS использует операционную систему и аппаратуру, то окажется, что все прикладные программы можно отнести к одной из следующих четырех категорий:

- Программы, совместимые на уровне MS-DOS
- Программы, совместимые на уровне ПЗУ BIOS
- Программы, совместимые на уровне аппаратуры
- «Некорректные» ("ill-behaved") программы

Программы, совместимые на уровне MS-DOS, используют только документированные вызовы функций MS-DOS и не вызывают ПЗУ BIOS, а также не обращаются непосредственно к аппаратуре. Такие программы используют для управления экраном управляющие Esc-последовательности (реализующие функции драйвера ANSI.SYS), а их ввод и вывод подчиняются операторам перенаправления. MS-DOS-совместимая программа будет выполняться на любой машине, поддерживающей MS-DOS, независимо от конфигурации этой машины. Из-за относительной примитивности встроенных в MS-DOS драйверов дисплея и последовательного порта в эту категорию попадает лишь малое число популярных программ, в основном – компиляторы, ассемблеры и компоновщики.

Программы, совместимые на уровне ПЗУ BIOS, используют документированные вызовы функций MS-DOS и ПЗУ BIOS, но не обращаются непосредственно к аппаратуре. Еще три года назад такой подход мог серьезно ограничить потенциальный рынок для программы. Сегодня, однако, доступность высококачественнных IBM-сов-

местимых микросхем ПЗУ с программами ПЗУ BIOS, выпускаемых такими компаниями, как Phoenix, обеспечила преобладание стандарта IBM на ПЗУ BIOS; фактически нет машины, на которой программа не могла бы столь же надежно опираться на интерфейс ПЗУ BIOS, как и интерфейс MS-DOS. Тем не менее, как уже отмечалось в гл. 6 и 7, драйверы дисплея и последовательного порта, включаемые в ПЗУ BIOS, все еще не отвечают требованиям высокопроизводительных интерактивных прикладных программ, поэтому популярных программ, относящихся к этой категории, тоже довольно мало.

Программы, совместимые на уровне аппаратуры, как правило, используют функции MS-DOS для управления массовой памятью, диспетчером памяти и другими подобными узлами, а при организации интерфейса с пользователем опираются на функции MS-DOS и ПЗУ BIOS, а также на непосредственные обращения к аппаратуре. Степень аппаратной зависимости таких программ может колебаться в очень широких пределах. Например, некоторые программы только записывают символы и атрибуты в буфер видеоконтроллера, а для переключения режимов и позиционирования курсора используют ПЗУ BIOS; другие полностью обходят видеодрайвер ПЗУ BIOS и целиком берут на себя управление видеоадаптером. К моменту написания этой книги большая часть популярных прикладных программ MS-DOS, повышающих производительность системы (текстовые процессоры, базы данных, телекоммуникационные программы и т. д.), попадала в эту категорию.

«Некорректными» называют такие прикладные программы, которые используют недокументированные функции MS-DOS и структуры данных, перехват прерываний MS-DOS или ПЗУ BIOS, а также непосредственные обращения к устройствам массовой памяти (в обход файловой системы MS-DOS). Такие программы обычно бывают крайне чувствительны к среде, в которой работают, и, как правило, должны «настраиваться» на каждую новую версию MS-DOS или модель PC. Фактически все популярные утилиты, остающиеся резидентными в памяти после завершения (terminate and stay resident, TSR), сетевые программы и ремонтные или оптимизационные дисковые пакеты попадают в эту категорию.

НАПИСАНИЕ КОРРЕКТНЫХ ПРИКЛАДНЫХ ПРОГРАММ ДЛЯ РАБОТЫ В СРЕДЕ MS-DOS

Ваш выбор путей решения конкретной задачи (функции MS-DOS, функции ПЗУ BIOS или непосредственное обращение к аппаратуре) должен всегда учитывать требования производительности; и, конечно, именно пользователь является окончательным судьей надежности и полезности программы. Тем не менее, желая разработать корректную прикладную программу, которая будет правильно выполняться и с будущими версиями MS-DOS, и с мультизадачными администраторами программ, выступающими как надстройки MS-DOS (например, Microsoft Windows), стоит придерживаться описанных ниже правил.

Структура программы

Стремитесь к тому, чтобы ваши программы представляли собой .EXE-файлы с отдельными сегментами программы, данных и стека; избегайте использования файлов .COM. Используйте соглашения фирмы Microsoft для имен сегментов и атрибутов, обсуждавшиеся в гл. 3. Для поиска оверлеев и файлов данных вашей программы используйте на этапе выполнения блок окружения; не «зашивайте» адреса каталогов в программу.

Анализ возможностей системы

В процессе инициализации вашей программы получите с помощью прерываний Int 21H с функцией 30H номер версии MS-DOS и удостоверьтесь в том, что все используемые вашей программой функции фактически доступны в этой версии. Если окажется, что рабочая версия MS-DOS неадекватна вашей программе, специально обратите внимание на то, с помощью каких функций выводятся аварийные сообщения и завершается задача.

Если ваша программа работает под управлением MS-DOS версий 3 и 4, используйте развитые средства этих систем. Например, открывая файл с помощью вызова Int 21H с функцией 3DH, можно указать режим совместного использования; с помощью вызова Int 21H с функциями 5AH и 5BH можно создавать временные или уникальные файлы; с помощью вызова Int 21H с функцией 59H можно получить расширенную информацию об ошибке (включая рекомендуемый способ восстановления). В разд. 2 этой книги для каждой функции MS-DOS приведены сведения об особенностях ее выполнения в разных версиях MS-DOS.

Ввод и вывод

Пользуйтесь исключительно функциями типа дескрипторов и проследите, чтобы поддержка полного пути распространялась на всю вашу программу (чтобы обеспечить пользователю возможность в процессе ввода имени файла указывать пути максимально допустимой длины). Всюду, где только возможно, используйте буферизацию ввода-вывода. Драйверы устройств MS-DOS версий 2.0 и выше могут работать со строками длиной до 64 Кбайт, и производительность программы возрастет, если вместо многих коротких записей вы будете записывать меньшее количество более длинных. Избегайте использовать FCB, функции Int 25H или Int 26H, а также драйвер диска ПЗУ BIOS. Если вам необходимо использовать блоки FCB, закрывайте их после отработки и не перемещайте в открытом состоянии. Избегайте открывать уже открытые блоки FCB и закрывать уже закрытые – эти, казалось бы, безобидные действия могут привести к неприятностям, если работает сетевое программное обеспечение.

Управление памятью

В процессе инициализации программы освободите всю память, которая не потребуется программе. (Это особенно важно для программ .COM.) Если вашей программе требуется дополнительная память для буферов и таблиц, распределяйте ее динамически по мере надобности и освобождайте, как только необходимость в ней отпадает. При наличии дополнительной памяти используйте ее, чтобы минимизировать запросы вашей программы на обладание обычной памятью.

Возьмите себе за правило не трогать память, не выделенную вашей программе. Для установки или анализа векторов прерываний пользуйтесь прерыванием Int 21H с функциями 25H и 35H; не редактируйте таблицу векторов прерываний непосредственно. Если вы изменяете содержимое векторов прерываний, сохраните их старые значения и восстановите их перед завершением программы.

Управление процессами

Чтобы сделать свою программу независимой от структуры PSP и информации о настройке адресов, при загрузке оверлеев или других программ используйте функцию

EXEC (Int 21H с функцией 4BH). Завершайте свою программу с помощью прерывания Int 21H с функцией 4CH, передавая код завершения, равный 0, если программа выполняется успешно, и ненулевой код, если возникли ошибки. Программа, родительская по отношению к вашей, может в этом случае проанализировать код возврата с помощью Int 21H с функцией 4DH или (в командном файле) с помощью предложения IF ERRORLEVEL.

Обработка исключительных состояний

Установите обработчики Ctrl-C (Int 23H) и критической ошибки (Int 24H), чтобы ваша программа не могла неожиданно завершиться при вводе пользователем Ctrl-C (Ctrl-Break) или при аппаратном отказе ввода-вывода. Это особенно важно, если ваша программа использует дополнительную память или устанавливает собственные обработчики прерываний.

ПЗУ BIOS И ПРИКЛАДНЫЕ ПРОГРАММЫ, СОВМЕСТИМЫЕ НА АППАРАТНОМ УРОВНЕ

Если вы чувствуете необходимость ради повышения производительности ввести в программу зависимость от ПЗУ BIOS или аппаратного обеспечения, старайтесь сконцентрировать эту зависимость в небольших, хорошо документированных процедурах, которые легко модифицировать при изменении аппаратуры. Используйте макрокоманды и другие аналогичные средства для скрытия характеристик аппаратуры и избегайте расползания «магических чисел» по всему телу программы.

Анализ возможностей системы

Если в вашей программе используются функции ПЗУ BIOS, вам следует уточнить в процессе выполнения программы модель компьютера, чтобы убедиться в фактическом наличии функций, требующихся программе. По адресу F000:FFFEH находится байт с идентификатором машины, в котором могут храниться следующие значения:

F8H	PS/2 Модели 70 и 80
F9H	PC Convertible
FAH	PS/2 Модель 30
FBH	PC/XT (поздние модели)
FCH	PC/AT, PC/XT-286, PS/2 Модели 50 и 60
FDH	PCjr
FEN	PC/XT (ранние модели)
FFN	PC "Classic"

В некоторых случаях могут идентифицироваться и подмодели; см. описаие Int 15H с функцией C0H. В разд. 3 этой книги для каждой фунции ПЗУ BIOS приведены сведения об особенностях ее выполнения с разными моделями машин.

При создании собственного видеодрайвера с непосредственным обращением к аппаратуре вы должны, в частности, определить тип и возможности видеоадаптера путем вызовов Int 10H, чтения из портов и анализа области данных ПЗУ BIOS по адресу 0040: 0000H и памяти, зарезервированной для ПЗУ BIOS EGA или VGA. Приемы выполнения этих проверок выходят за рамки нашей книги; они подробно описаны в справочном пособии Programmer's Guide to PC and PS/2 Video Systems (Microsoft Press, 1987).

Вопросы аппаратной совместимости

Некоторые элементы архитектуры персональных компьютеров IBM оказались замечательно стабильными и использовались, начиная от первых IBM PC на базе микропроцессора 8088 с частотой 4,77 МГц и до сегодняшних PS/2, модель 80 на базе микропроцессора 80386 с частотой 20 МГц. Послужной список фирмы IBM в совместимости снизу вверх ее видеоконтроллеров и контроллеров последовательной связи просто превосходен; во многих случаях аппаратно-зависимые программы, написанные для первых IBM PC, прекрасно работают на IBM PS/2, модель 80. Другие области относительной стабильности аппаратуры:

- управление звуком через порт 61H,
- микросхема таймера 8253, каналы 0 и 2 (порты 40H, 42H и 43H),
- адаптер игр (порт 201H),
- управление системой прерываний с помощью регистра маски контроллера прерываний 8259 через порт 21H.

Тем не менее непосредственная генерация звука и манипуляции с таймером 8253 или контроллером прерываний 8259 могут привести к неприятностям, если ваша программа работает в среде администратора мультизадачного режима, например Microsoft Windows или DesqView.

Отображение клавиатуры, контроллер клавиатуры, а также контроллеры гибкого и жесткого дисков являются примерами относительно нестабильной аппаратуры. Программы, работающие с клавиатурой или дисками в обход MS-DOS, вряд ли будут функционировать одинаково правильно на разных моделях PC; кроме того, они скорее всего будут вступать в конфликт друг с другом или с корректными программами.

СОВМЕСТИМОСТЬ С OS/2

MS-DOS во многих отношениях совместима снизу вверх с OS/2 – мультизадачной операционной системой защищенного режима и виртуальной памяти, разработанной фирмой Microsoft для компьютеров с микропроцессорами 80286 и 80386. Графический интерфейс пользователя (администратор представления) практически идентичен Microsoft Windows 2.0. Система OS/2 версий 1.0 и 1.1 использует в точности тот же формат диска, что и MS-DOS, так что эти системы могут легко обмениваться файлами. Более того, OS/2 включает модуль, носящий имя «Среда совместимости с DOS» или 3.x Box, который позволяет выполнять одну прикладную программу MS-DOS параллельно с прикладными программами защищенного режима OS/2.

3.x Box захватывает вызовы функций Int 21H и отображает их на функциональные вызовы OS/2, эмулируя среду MS-DOS 3.3 с загруженным модулем разделения файлов (SHARE.EXE), однако возвращая в ответ на прерывание Int 21H с функцией 30H основной номер версии 10 вместо 3. Модуль 3.x Box также поддерживает большинство вызовов ПЗУ BIOS, либо эмулируя их функции, либо путем запирания устройства и вызова истинной программы ПЗУ BIOS. Кроме того, 3.x Box поддерживает область данных ПЗУ BIOS, выдает прикладным программам такты таймера через Int 1CH и поддерживает некоторые недокументированные средства MS-DOS и структуры данных, так что большинство утилит TSR могут функционировать обычным образом. Тем не менее эмуляция модулем 3.x Box средств MS-DOS несовершенна, и следует иметь в виду наличие некоторых ограничений при выполнении программ для MS-DOS под управлением OS/2.

Важнейшее ограничение, накладываемое на прикладные программы MS-DOS, заключается в том, что, когда эти программы находятся в фоновом состоянии (background), они не получают циклов центрального процессора. Другими словами,

если «выбрана» прикладная программа защищенного режима, что дает право пользователю взаимодействовать с ней, прикладная программа MS-DOS «замораживается». Если же программа захватила какие-то векторы прерываний (например, последовательного порта или тактов таймера), то эти прерывания не будут обслуживаться до тех пор, пока программа не будет снова выбрана и не перейдет в приоритетное состояние. OS/2 вынуждена замораживать прикладные программы MS-DOS, находящиеся в фоновом состоянии, потому что они выполняются в реальном режиме и, следовательно, на них не распространяется аппаратная защита памяти; в результате нет гарантий, что они не вступят в конфликт с процессом защищенного режима, под контролем которого находятся экран и клавиатура.

В 3.x Box ограничивается использование FCB, как это имеет место и в MS-DOS версии 3 или 4 с загруженным модулем SHARE.EXE. Нельзя открыть файл с FCB, если его использует какой-то другой процесс. Число FCB, которые могут быть открыты одновременно, не может быть больше 16 или числа, указанного в директиве FCBS-файла CONFIG.SYS. Даже если используются функции типа дескрипторов, эти функции могут неожиданно давать отказы в результате активности других процессов (например, если процесс защищенного режима уже открыл файл в режиме разделения с «полным запретом»); большинство прикладных программ MS-DOS не предназначены для работы в режиме разделения файлов и не могут аккуратно обрабатывать такого рода ошибки.

Операции непосредственной записи на жесткий диск с помощью прерываний Int 26H или Int 13H запрещены. Это предохраняет от разрушения файловую систему, учитывая, что прикладные программы защищенного режима могут записывать на тот же диск. Можете представить себе, какая будет путаница, если типичная утилита MS-DOS восстановления стертых файлов начнет изменять корневой каталог и FAT в то же самое время, когда программа базы данных защищенного режима модифицирует свои файлы и индексы!

Прикладные программы MS-DOS, которые пытаются перепрограммировать контроллер 8259 с целью перемещения таблицы векторов прерываний или модифицировать векторы прерываний, уже принадлежащие драйверу устройства OS/2, завершаются операционной системой. Прикладные программы MS-DOS могут изменять регистр маски прерываний 8259, запрещать и снова разрешать прерывания на свой страх и риск и читать или писать через любые порты ввода-вывода. Очевидное следствие всего этого – программа MS-DOS, выполняемая в среде 3.x Box, может в любой момент стать причиной полной аварии системы OS/2; такую цену мы платим за саму возможность запуска прикладных программ реального режима.

ПЕРЕНОС ПРИКЛАДНЫХ ПРОГРАММ MS-DOS в OS/2

Интерфейс прикладных программ (Application program interface, API), предоставляемый системой OS/2 программам защищенного режима, заметно отличается от знакомого нам интерфейса MS-DOS с модулем 3.x Box OS/2, реализуемого с помощью прерываний Int 21H. Тем не менее API OS/2 функционально является законным подмножеством MS-DOS. Это упрощает преобразование корректных прикладных программ MS-DOS для их запуска в защищенном режиме, где они могут использовать преимущества OS/2: виртуальную память, многозадачный режим и средства межпроцессных взаимодействий.

Чтобы дать вам представление о сущности интерфейса прикладных программ OS/2 и одновременно о средствах, которых следует избегать в программировании для MS-DOS, если желательно обеспечить переносимость в OS/2, я опишу свою собственную методику преобразования для среды OS/2 существующих программ, написанных

для MS-DOS на языке ассемблера. Из методических соображений я разделил процесс преобразования на пять шагов и каждому присвоил легко запоминающееся звучное имя:

1. Сегментация.
2. Рационализация.
3. Капсулирование.
4. Преобразование.
5. Оптимизация.

Первые три шага могут (и должны) тестироваться в среде MS-DOS; только два последних требуют наличия OS/2 и инструментальных средств защищенного режима. Знакомясь с материалом, вы увидите, что программа для MS-DOS, написанная в соответствии с приведенными выше указаниями, направленными на обеспечение переносимости, потребует относительно небольших усилий для переноса в среду защищенного режима. Это естественное следствие работы «рука об руку» с операционной системой вместо попыток борьбы с нею.

Сегментация

Большая часть возможностей защищенного режима МП 80286 связана с новым способом адресации памяти. В реальном режиме МП 80286 существенным образом эмулирует процессоры 8088/86, и содержимое регистра сегмента непосредственно соответствует физическому адресу памяти. MS-DOS работает на МП 80286 в реальном режиме.

Если МП 80286 используется в защищенном режиме, как это имеет место в среде OS/2, в адресацию памяти добавляется дополнительный уровень косвенности[1]. Регистр сегмента содержит селектор, который представляет собой индекс в таблице дескрипторов[2]. Дескриптор определяет физический адрес и длину сегмента памяти его характеристики (выполнимый сегмент данных только для чтения или сегмент данных для чтения-записи) и права доступа, а также является ли сегмент в текущий момент резидентным в ОЗУ памяти или выгружен на диск. Каждый раз, когда программа загружает регистр сегмента или обращается к памяти, аппаратура МП 80286 проверяет соответствующий дескриптор и уровень привилегий программы, генерируя отказ, если селектор или операция обращения к памяти оказываются недопустимыми. Отказ действует аналогично аппаратному прерыванию, позволяя операционной системе получить управление и предпринять соответствующие действия.

Такой способ адресации памяти в защищенном режиме приводит к двум важным для прикладных программ следствиям. Во-первых, прикладная программа уже не может выполнять арифметические действия над содержимым регистров сегментов (потому что селекторы представляют собой магические числа и не имеют непосредственного отношения к физическим адресам памяти) или использовать регистры сегментов для хранения промежуточных данных. Программа не должна загружать регистр сегмента чем-либо кроме законного селектора, образованного загрузчиком OS/2 или полученного в результате вызова функции OS/2 распределения памяти. Во-вторых, программа должна строго отделять машиные коды («текст») от данных,

---

[1] Хотя МП 80386 присущи дополнительные режимы и возможности адресации, используемые версии OS/2 работают с МП 80386, как если бы это был МП 80286.

[2] Здесь слово «дескриптор» выступает как перевод английского «descriptor», в то время как в остальном тексте книги «дескриптор» (файла или устройства) соответствует английскому «handle». – *Прим. перев.*

помещая их в отдельные сегменты с различными селекторами (потому что выполнимый селектор является запрещенным для записи, и наоборот).

Соответственно первый шаг в процессе перевода программы в среду OS/2 заключается в преобразовании ее в программу .EXE-типа, которая использует описанные в гл. 3 соглашения фирмы Microsoft относительно сегмента, класса и группы. Как минимум в программу должны входить один сегмент программы и один сегмент данных; кроме того, программа должна объявить группу (со специальным именем DGROUP), содержащую сегмент «ближних» данных, стек и локальную динамическую область («кучу»), если таковые имеют место. В то же время вы обязаны удалить или переписать любые фрагменты программы, которые выполняют непосредственные преобразования содержимого сегментов.

После структурирования и сегментации заново ассемблируйте и скомпонуйте вашу программу и проверьте, работает ли она по-прежнему в среде MS-DOS. Изменение и добавление сегментации часто выявляют в программе скрытые допущения относительно адресации, поэтому лучше всего сначала обнаружить и устранить все дефекты такого рода и лишь потом перейти к последующим изменениям в программе.

Рационализация

После того как вы успешно сегментировали вашу программу и она может быть скомпонована и выполнена как файл .EXE в среде MS-DOS, можно приступить к следующему шагу – рационализации программы. Под рационализацией я понимаю преобразование вашей программы в абсолютно корректную прикладную программу для MS-DOS.

Прежде всего вы должны полностью устранить все строки непосредственного обращения к адаптерам периферийных устройств, изменения приоритетов прерываний, редактирования системной таблицы векторов прерываний, а также фрагменты, использующие скорость или другие характеристики ЦП (например, программные циклы, осуществляющие фиксированную временную задержку). В защищенном режиме управление системой прерываний целиком возлагается на операционную систему и системные драйверы устройств, порты ввода-вывода могут использоваться прикладной програмой для записи или чтения только при строго определенных условиях, а циклы программной задержки потребляют время ЦП, которое могло быть отдано другим процессам.

Как уже упоминалось ранее, программы вывода на экран являются типичными в том отношении, что они обычно несут черты аппаратной зависимости. Непосредственное управление видеоадаптером и его буфером вызывает очевидные трудности при работе в среде многозадачной системы с защитой памяти, какой является OS/2. С целью обеспечения переносимости программы вы должны преобразовать все фрагменты вывода текста на дисплей, модификации атрибутов символов и изменения формы или положения курсора в вызовы Int 21H с функцией 40H, используя Esc-последовательности ANSI, либо в вызовы Int 10H ПЗУ BIOS. Точно так же все аппаратно-зависимые операции с клавиатурой должны быть преобразованы в вызовы Int 21H с функцией 3FH или Int 16H ПЗУ BIOS.

После удаления из вашей программы всей аппаратной зависимости вы должны обеспечить ее корректность в плане использования системной памяти. В среде MS-DOS прикладной программе обычно передается вся оставшаяся в системе память, и прикладная программа может использовать ее по своему усмотрению; в среде OS/2 справедливо обратное: процессу первоначально распределяется лишь столько памяти, сколько необходимо для размещения программных строк, объявленных полей данных и стека. Вы можете заставить загрузчик MS-DOS функционировать подобно загрузчику

OS/2, указав при компоновке программы ключ/CPARMAXALLOC. Либо можно с помощью прерывания Int 21H с функцией 4AH освободить всю лишнюю память в процессе инициализации, как это рекомендовалось ранее.

После того как программа выполнит все действия по инициализации, следует динамически получить и освободить дополнительную память, которая может потребоваться программе для буферов и таблиц. Это делается с помощью прерываний MS-DOS Int 21H с функциями 48H и 49H. Для обеспечения совместимости с защищенным режимом ограничьте размер любого одиночного распределяемого блока величиной 65536 байт или меньше, несмотря на то что MS-DOS позволяет распределять блоки большего размера.

Наконец, следует позаботиться о работе с файлами и устройствами. Замените все файловые функции типа FCB на эквивалентные им функции типа дескрипторов, поскольку OS/2 в защищенном режиме не поддерживает FCB. Проверьте правильность обозначений путей; имейте в виду, что MS-DOS и 3.x Box молчаливо усекают имя или расширение, а OS/2 не может открыть или создать файл в защищенном режиме, если имя или расширение оказываются слишком длинными; в этом случае возвращается ошибка. Замените использование определенных заранее дескрипторов устройств (стандартный дополнительный порт и стандартная печать) на явные операции открытия COM1, PRN, LPT1 и т.д., используя для операций чтения и записи полученные таким образом дескрипторы. OS/2 не предоставляет процессам стандартные дескрипторы последовательного порта и принтера.

Капсулирование

Если вы достигли этого места, имея на руках корректную сегментированную прикладную программу для среды MS-DOS, то худшая часть работы по обеспечению переносимости в среду OS/2 позади. Теперь вы готовы к преобразованию программы для выполнения в защищенном режиме путем «капсулирования» в отдельных подпрограммах частей, специфичных для операционной системы. Цель этого преобразования заключается в локализации «знания» программой операционной среды в небольших процедурах, которые затем несложно модифицировать, не затрагивая оставшуюся часть программы.

В качестве примера капсулирования рассмотрим типичный запрос прикладной программы MS-DOS на запись строки на стандартном устройстве вывода (рис. 16.1). Для облегчения преобразования в среду OS/2 вам следует поместить каждый такой запрос на запись в файл или устройство в небольшую подпрограмму, которая «прячет» детали фактического обращения к операционной системе, как это показано на рис. 16.2.

Примером другой операции, требующей капсулирования и не обязательно содержащей вызов системной функции, является фрагмент прикладной программы с обращением к параметрам командной строки, переменным блока окружения и имени файла, из которого он был загружен. В среде MS-DOS эта информация, как было показано в гл. 3 и 12, частично содержится в префиксе программного сегмента (PSP), а частично – в блоке окружения; в среде OS/2 нет такой вещи, как PSP, и имя файла программы вместе с информацией, введенной в командной строке, добавляется к блоку окружения.

После завершения капсулирования системных средств и доступа к PSP и окружению еще раз подвергните свою программу тщательному тестированию в среде MS-DOS. Это ваш последний шанс, пока вы работаете в знакомой среде и можете воспользоваться привычными средствами отладки, выловить тонкие ошибки, которые вы могли внести в программу в процессе выполнения трех описанных выше шагов.

```
stdin   equ     0               ; Дескриптор стандартного ввода
stdout  equ     1               ; Дескриптор стандартного вывода
stderr  equ     2               ; Дескриптор стандартной ошибки

msg     db      'Это пробное сообщение'
msg_len equ     $-msg

        .
        .
        .
        mov     dx,seg msg      ; DS:DX = адрес сообщения
        mov     ds,dx
        mov     dx,offset DGROUP:msg
        mov     cx,msg_len      ; CX = длина сообщения
        mov     bx,stdout       ; BX = дескриптор
        mov     ah,40h          ; AH = функция 40h, записать
        int     21h             ; Переход в MS-DOS
        jc      error           ; Переход, если ошибка
        cmp     ax,msg_len      ; Все символы записаны?
        jne     diskfull        ; Нет, устройство заполнено
        .
        .
        .
```

Рис. 16.1. Типичный фрагмент вызова функции MS-DOS.

Конкретная последовательность, приведенная на рисунке, записывает строку на стандартное устройство вывода. Поскольку стандартный вывод может быть без ведома программы перенаправлен в файл, в процессе вызова следует убедиться, что все запрошенные символы действительно были выведены; если возвращенная длина меньше запрошенной, то обычно это указывает на то, что стандартный вывод был перенаправлен в файл на диске, а места на диске нет.

```
stdin   equ     0               ; Дескриптор стандартного ввода
stdout  equ     1               ; Дескриптор стандартного вывода
stderr  equ     2               ; Дескриптор стандартной ошибки

msg     db      'Это пробное сообщение'
msg_len equ     $-msg
        .
        .
        .
        mov     dx,seg msg      ; DS:DX = адрес сообщения
        mov     ds,dx
        mov     dx,offset DGROUP:msg
        mov     cx,msg_len      ; CX = длина сообщения
        mov     bx,stdout       ; BX = дескриптор
        call    write           ; Выполнить запись
        jc      error           ; Переход, если ошибка
        cmp     ax,msg_len      ; Все символы записаны?
        jne     diskfull        ; Нет, устройство заполнено
        .
        .
        .

write   proc    near            ; Запись в файл или устройство
```

Рис. 16.2. Фрагмент программы из рис. 16.1 после «капсулирования».

Часть программы, зависящая от используемой операционной системы, изолирована путем помещения в подпрограмму, которая вызывается из других точек прикладной программы.

```
                                ; При вызове:
                                ; BX = дескриптор
                                ; CX = длина данных
                                ; DS:DX = адрес данных
                                ; При возврате:
                                ; Если выполнение успешное, перенос сброшен
                                ; и AX = число записанных байтов
                                ; Если ошибка, перенос установлен
                                ; и AX = код ошибки

        mov     ah,40h          ; Функция 40h = записать
        int     21h             ; Переход в MS-DOS
        ret                     ; Состояние возвращается в CY и AX

write   endp
        .
        .
        .
```

Рис. 16.2.

Преобразование

В качестве следующего шага вам надлежит переписать все системно-зависимые процедуры, созданные вами на шаге капсулирования так, чтобы они соответствовали API защищенного режима OS/2. В отличие от функций MS-DOS, активизируемых с помощью программных прерываний и получающих параметры через регистры, функции API OS/2 запрашиваются с помощью дальнего вызова именованной точки входа. Параметры передаются на стеке вместе с адресами переменных в сегменте данных вызывающей программы, которые получают какие-либо результаты, возвращаемые функцией. Состояние операции возвращается в регистре AX – 0 при успешном завершении функции и в код ошибки – в противном случае. Все остальные регистры сохраняются.

Хотя в мою задачу не входит изложение подробного введения в программирование в среде OS/2, на рис. 16.3 показан окончательный вид предыдущего примера после

```
stdin   equ     0               ; Дескриптор стандартного ввода
stdout  equ     1               ; Дескриптор стандартного вывода
stderr  equ     2               ; Дескриптор стандартной ошибки

        extrn   DosWrite:far

msg     db      'Это пробное сообщение'
msg_len equ     $-msg

wlen    dw      ?               ; Получает число реально
                                ;записанных байтов
```

Рис. 16.3. Программа из рис. 16.2 после «преобразования».

Вызов функции MS-DOS заменен эквивалентным функциональным вызовом OS/2. Поскольку системно-зависимые строки были спрятаны внутри подпрограммы на предыдущем шаге капсулирования, запросы на операции записи из других мест программы не изменяются. Заметьте, что функцию OS/2 пришлось объявить как внешнее имя с атрибутом «дальний», а в сегмент данных прикладной программы была добавлена переменная WLEN, принимающая число фактически записанных файлов.

```
        .
        .
        mov     dx,seg msg      ; DS:DX = адрес сообщения
        mov     ds,dx
        mov     dx,offset DGROUP:msg
        mov     cx,msg_len      ; CX = длина сообщения
        mov     bx,stdout       ; BX = дескриптор
        call    write           ; Выполнить запись
        jc      error           ; Переход, если ошибка
        cmp     ax,msg_len      ; Все символы записаны?
        jne     diskfull        ; Нет, устройство заполнено
        .
        .
        .

write   proc    near            ; Запись в файл или на устройство
                                ; При вызове:
                                ; BX = дескриптор
                                ; CX = длина данных
                                ; DS:DX = адрес данных
                                ; При возврате:
                                ; Если выполнение успешное, перенос сброшен
                                ; и AX = число записанных байтов
                                ; Если ошибка, перенос установлен
                                ; и AX = код ошибки

        push    bx              ; Дескриптор
        push    ds              ; Адрес данных
        push    dx
        push    cx              ; Длина данных
        push    ds              ; Получает записанную длину
        mov     ax,offset DGROUP:wlen
        push    ax
        call    DosWrite        ; Переход в OS/2
        or      ax,ax           ; Запись была успешной?
        jnz     write1          ; Переход, запись не выполнилась
        mov     ax,wlen         ; Ошибок нет, команда OR очистила CY,
        ret                     ; а AX := числозаписанных байтов

write1: stc                     ; Ошибка записи, возврат с установленным CY,
        ret                     ; а AX = номер ошибки

write   endp
        .
        .
        .
```

Рис. 16.3.

преобразования его в OS/2. Обратите внимание на добавление предложения extrn, переменной wlen и эмулирования состояния функции MS-DOS. Приведенная программа, возможно, не отличается изяществом, но она служит задаче локализации необходимых изменений на очень небольшом участке исходного файла. Некоторые функции OS/2 (например, DosOpen) требуют параметров, не имеющих аналогов в MS-DOS; обычно вы имеете возможность выбрать разумные значения для этих дополнительных параметров, что сделает их существование незаметным для остальной части прикладной программы.

На рис. 16.4, 16.5 и 16.6 приведены функции OS/2, эквивалентные некоторым средствам MS-DOS и ПЗУ BIOS (вызовы Int 21H, Int 10H и Int 16H). Функции MS-DOS, связанные с FCB и PSP, не включены в эти таблицы, потому что OS/2 не поддерживает ни ту ни другую структуру. TSR-функции MS-DOS также опущены. Поскольку OS/2 является истинно многозадачной системой, процесс не нуждается в завершении, чтобы остаться резидентным на время выполнения другого процесса.

MS-DOS *Функция Int 21H*	**Описание**	**Функция OS/2**
0	Завершить процесс	DosExit
1	Ввод символа с эхо	KbdCharIn
2	Вывод символа	VioWrtTTY
3	Ввод с дополнительного устройства	DosRead
4	Вывод на дополнительное устройство	DosWrite
5	Вывод на принтер	DosWrite
6	Непосредственный ввод-вывод через консоль	KbdCharIn, VioWrtTTY
7	Нефильтрованный ввод без эхо	KbdCharIn
8	Ввод символа без эхо	KbdCharIn
9	Вывести на экран строку	VioWrtTTY
0AH(10)	Буферизованный ввод с клавиатуры	KbdStringIn
0BH(11)	Проверить состояние ввода	KbdPeek
0CH(12)	Сбросить буфер и ввести	KbdFlushBuffer, KbdCharIn
0DH(13)	Привести диск в исходное состояние	DosBufReset
0EH(14)	Выбрать диск	DosSelectDisk
19H(25)	Получить текущий диск	DosQCurDisk
1BH(27)	Получить информацию о дисководе, используемом по умолчанию	DosQFSInfo
1CH(28)	Получить информацию о дисководе	DosQFSInfo
2AH(42)	Получить дату	DosGetDateTime
2BH(43)	Установить дату	DosSetDateTime
2CH(44)	Получить время	DosGetDateTime
2DH(45)	Установить время	DosSetDateTime
2EH(46)	Установить флаг верификации	DosSetVerify
30H(48)	Получить версию MS-DOS	DosGetVersion
36H(54)	Получить информацию о характеристиках дисковода	DosQFSInfo
38H(56)	Получить или установить национальную информацию	DosGetCtryInfo
39H(57)	Создать каталог	DosMkdir
3AH(58)	Удалить каталог	DosRmdir
3BH(59)	Установить текущий каталог	DosChdir
3CH(60)	Создать файл	DosOpen
3DH(61)	Открыть файл	DosOpen
3EH(62)	Закрыть файл	DosClose
3FH(63)	Прочитать из файла или устройства	DosRead
40H(64)	Записать в файл или на устройство	DosWrite
41H(65)	Удалить файл	DosDelete
42H(66)	Установить указатель файла	DosChgFilePtr
43H(67)	Получить или установить атрибуты файла	DosQFileMode, DosSetFileMode
44H(68)	Управление вводом-выводом (IOCTL)	DosDevIOCtl
45H(69)	Дублировать дескриптор	DosDupHandle
46H(70)	Перенаправить дескриптор	DosDupHandle
47H(71)	Получить текущий каталог	DosQCurDir

Рис. 16.4. Таблица некоторых функций MS-DOS и их аналогов в OS/2.

Заметьте, что функции OS/2 обычно отличаются большей мощностью и гибкостью, чем соответствующие им функции MS-DOS, и что приведенный список не исчерпывает средства OS/2.

MS-DOS	Описание	Функция OS/2
48H(72)	Выделить блок памяти	DosAllocSeg
49H(73)	Освободить блок памяти	DosFreeSeg
4AH(74)	Изменить размер блока памяти	DosReAllocSeg
4BH(75)	Выполнить программу	DosExecPgm
4CH(76)	Завершить процесс с кодом возврата	DosExit
4DH(77)	Получить код возврата	DosCWait
4EH(78)	Найти первый файл	DosFindFirst
4FH(79)	Найти следующий файл	DosFindNext
54H(84)	Получить флаг верификации	DosQVerify
56H(86)	Переименовать файл	DosMove
57H(87)	Получить или установить дату и время создания файла	DosQFileInfo, DosSetFileInfo
59H(89)	Получить расширенную информацию об ошибке	DosErrClass
5BH(91)	Создать новый файл	DosOpen
5CH(92)	Запереть или отпереть район файла	DosFileLocks
65H(101)	Получить расширенную национальную информацию	DosGetCtryInfo
66H(102)	Получить или установить кодовую страницу	DosGetCp, DosSetCp
67H(103)	Установить число дескрипторов	DosSetMaxFH
68H(104)	Сбросить в файл	DosBufReset
6CH(108)	Расширенное открытие файла	DosOpen

Рис. 16.4.

ПЗУ BIOS	Описание	Функция OS/2
Функция Int 10H		
0	Выбрать видеорежим	VioSetMode
1	Установить тип курсора	VioSetCurType
2	Установить позицию курсора	VioSetCurPos
3	Получить позицию курсора	VioGetCurPos
6	Инициализировать или прокрутить окно вверх	VioScrollUp
7	Инициализировать или прокрутить окно вниз	VioScrollDn
8	Прочитать символ и атрибут	VioReadCellStr
9	Записать символ и атрибут	VioWrtNCell
0AH(10)	Записать символ	VioWrtNChar
0EH(14)	Записать символ в режиме телетайпа	VioWrtTTY
0FH(15)	Получить видеорежим	VioGetMode
10H(16)	Установить палитру, цвет края и т. д.	VioSetState
13H(19)	Записать строку в режиме телетайпа	VioWrtTTY

Рис. 16.5. Таблица функций Int 10H ПЗУ BIOS для работы с драйвером видеотерминала, используемых прикладными программами в MS-DOS, и их эквиваленты в OS/2. Это неполный список видеосредств OS/2.

ПЗУ BIOS	Описание	Функция OS/2
Функция Int 16H		
0	Прочитать символ с клавиатуры	KbdCharIn
1	Получить состояние клавиатуры	KbdPeek
2	Получить флаги клавиатуры	KbdGetStatus

Рис. 16.6. Таблица функций Int 16H ПЗУ BIOS для работы с драйвером клавиатуры, используемых прикладными программами в MS-DOS, и их эквиваленты в OS/2. Это неполный список средств обслуживания клавиатуры OS/2.

Оптимизация

Теперь, когда ваша программа работает в защищенном режиме, наступило время дать пояснения по поводу некоторых изменений, выполненных в процессе преобразования, и поговорить о методах оптимизации. Представляют интерес три основные категории:

1. Модификация программных строк интерфейса с пользователем с целью привлечения более мощных функций клавиатуры и дисплея программного интерфейса (API) OS/2.
2. Введение, где возможно, специфичных для МП 80286 машинных команд.
3. Переделка прикладной программы с целью использования средств OS/2, характерных только для защищенного режима. (Естественно, прикладная программа автоматически выигрывает от наличия в OS/2 средств виртуальной памяти; программа может распределять память до тех пор, пока не истощится физическая память и пространство свопинга на диске.)

Модификация подпрограмм, в которых капсулированы ввод и вывод пользователя, с целью использования преимуществ дополнительных средств OS/2 не представляет особого труда, однако в результате можно получить поистине драматическое повышение производительности. Например, видеодрайвер OS/2 предоставляет набор средств, существенно более совершенных, чем поддержка экрана в MS-DOS и ПЗУ BIOS. Сюда относятся скоростной вывод строк и атрибутов в любом месте экрана, «чтение назад» выбранных областей с экрана в буфер, а также прокрутка во всех четырех направлениях.

Машинные команды, специфичные для МП 80286, могут оказать существенную помощь в сокращении размера программы и увеличении скорости ее выполнения. К наиболее полезным командам можно отнести логические и циклические сдвиги на определенное число битов, отличное от 1, трехоперандное умножение, в котором один из операндов представляет собой непосредственое значение, и команду сохранения на стеке непосредственного значения (последняя команда особенно удобна при вызовах функций OS/2). Например, последовательность команд из программы на рис. 16.3

```
mov ax, offset DGROUP:wlen
push ax
```

можно заменить одной командой

```
push offset DGROUP:wlen
```

Переделка прикладной программы с целью использования в полной мере защищенного режима OS/2 требует внимательного изучения как самой программы, так и API OS/2, но это изучение может принести плоды в виде заметного роста производительности, простоты обслуживания и удобств совместного использования программы несколькими процессами. Например, часто различные участки прикладной программы имеют дело с устройствами ввода-вывода, работающими с существенно разной скоростью: клавиатурой, диском и экраном. Программа упрощается при одновременном улучшении ее характеристик, если выделить указанные элементы в подпроцессы (называемые в OS/2 нитями, threads), выполняемые асинхронно, взаимодействующие через разделяемые ими структуры данных и синхронизуемые друг с другом в случае необходимости с помощью семафоров.

В качестве другого примера можно привести ситуацию, когда несколько прикладных программ тесно связаны и содержат много идентичных или весьма схожих

процедур. OS/2 дает возможность сконцентрировать эти процедуры в библиотеке динамической компоновки (dynamic link library). Подпрограммы этой библиотеки подсоединяются к программе на этапе загрузки (а не с помощью компоновщика LINK, как это делается традиционно) и используются совместно всеми нуждающимися в них процессами. Такая методика уменьшает размер каждого прикладного файла .EXE и приводит к более эффективному использованию памяти. Кроме того, применение библиотек динамической компоновки существенно упрощает сопровождение программ; подпрограммы библиотеки могут в любое время модифицироваться и отлаживаться, а использующие их прикладные программы автоматически улучшают свои характеристики при очередном запуске.

Раздел 2. Справочник по функциям MS-DOS

Пояснения для читателя

В этом разделе описывается сервис, предоставляемый ядром системы MS-DOS прикладным программам через системные прерывания 20H – 2FH. Все функции MS-DOS представлены по единой схеме:

- Заголовок, содержащий имя файла, номер системного прерывания и номер функции, а также пиктограмму с указанием версии MS-DOS, в которую впервые введена данная функция. Можно считать, что функция имеется во всех последующих версиях MS-DOS, если не оговорено особо.
- Краткий обзор действий, осуществляемый функцией, и типичные ситуации, где целесообразно ее использование.
- Описание аргументов функции.
- Результаты и/или индикаторы ошибок, возвращаемые функцией. В описании прерывания Int 21H с функцией 59H можно найти обширную таблицу с кодами ошибок.
- Замечания, которые описывают специальное использование или особенности функции.
- Пример, содержащий схему программы на языке ассемблера, где демонстрируется применение функции.

Пиктограммы с номером версии, используемые в разделах обзора аргументов, результатов и замечаний, относятся к конкретной указанной версии или подверсии, если только в них не включен знак «+», обозначающий данную и все последующие версии.

В примерах программы для ясности могут содержаться команды, включение которых в реальную программу необязательно. Так, в большинстве примеров программы при передаче системе MS-DOS адреса имени файла или буфера явно устанавливают сегментные регистры; в реальных прикладных программах обычно сегментные регистры инициализируются только один раз при входе в программу, а потом их содержимое не изменяется.

Список функций прерывания Int 21H по номерам

Hex	Dec	Описание	Версия	F/H[1)]
00H	0	Завершить процесс	1.0+	
01H	1	Ввод символа с эхо	1.0+	
02H	2	Вывод символа	1.0+	
03H	3	Дополнительный ввод	1.0+	
04H	4	Дополнительный вывод	1.0+	
05H	5	Вывод на принтер	1.0+	
06H	6	Непосредственный ввод-вывод через консоль	1.0+	
07H	7	Нефильтрованный ввод без эхо	1.0+	

Продолжение таблицы

Hex	Dec	Описание	Версия	F/H[1)]
08H	8	Ввод символа без эхо	1.0+	
09H	9	Вывести на экран строку	1.0+	
0AH	10	Буферизованный ввод с клавиатуры	1.0+	
0BH	11	Проверить состояние ввода	1.0+	
0CH	12	Сбросить буфер и затем ввести	1.0+	
0DH	13	Сбросить диск	1.0+	
0EH	14	Выбрать диск	1.0+	
0FH	15	Открыть файл	1.0+	F
10H	16	Закрыть файл	1.0+	F
11H	17	Найти первый файл	1.0+	F
12H	18	Найти следующий файл	1.0+	F
13H	19	Удалить файл	1.0+	F
14H	20	Последовательное чтение	1.0+	F
15H	21	Последовательная запись	1.0+	F
16H	22	Создать файл	1.0+	F
17H	23	Переименовать файл	1.0+	F
18H	24	Зарезервирована		
19H	25	Получить текущий диск	1.0+	
1AH	26	Установить адрес DTA	1.0+	
1BH	27	Получить информацию о дисководе, действующем по умолчанию		
1CH	28	Получить информацию о дисководе	2.0+	
1DH	29	Зарезервирована		
1EH	30	Зарезервирована		
1FH	31	Зарезервирована		
20H	32	Зарезервирована		
21H	33	Чтение в режиме прямого доступа	1.0+	F
22H	34	Запись в режиме прямого доступа	1.0+	F
23H	35	Получить размер файла	1.0+	F
24H	36	Установить номер относительной записи	1.0+	F
25H	37	Установить вектор прерываний	1.0+	
26H	38	Создать новый PSP	1.0+	
27H	39	Чтение блока в режиме прямого доступа	1.0+	F
28H	40	Запись блока в режиме прямого доступа	1.0+	F
29H	41	Выделить и перенести в FCB имя файла	1.0+	
2AH	42	Получить дату	1.0+	
2BH	43	Установить дату	1.0+	
2CH	44	Получить время	1.0+	
2DH	45	Установить время	1.0+	
2EH	46	Установить флаг проверки	1.0+	
2FH	47	Получить адрес DTA	2.0+	
30H	48	Получить номер версии MS-DOS	2.0+	
31H	49	Завершить и оставить резидентной	2.0+	
32H	50	Зарезервирована		
33H	51	Получить или установить флаг BREAK, получить код загрузочного дисковода	2.0+	
34H	52	Зарезервирована		
35H	53	Получить вектор прерываний	2.0+	
36H	54	Получить информацию о характеристиках дисковода		
37H	55	Зарезервирована		
38H	56	Получить или установить информацию о национальном формате	2.0+	
39H	57	Создать каталог	2.0+	
3AH	58	Удалить каталог	2.0+	
3BH	59	Установить текущий каталог	2.0+	
3CH	60	Создать файл	2.0+	H
3DH	61	Открыть файл	2.0+	H
3EH	62	Закрыть файл	2.0+	H
3FH	63	Прочитать файл или устройство	2.0+	H

Продолжение таблицы

Hex	Dec	Описание	Версия	F/H[1]
40H	64	Записать в файл или на устройство	2.0+	H
41H	65	Удалить файл	2.0+	H
42H	66	Установить указатель файла	2.0+	H
43H	67	Получить или установить атрибуты файла	2.0+	
44H	68	Управление вводом-выводом (IOCTL)	2.0+	
45H	69	Дублировать дескриптор	2.0+	
46H	70	Перенаправить дескриптор	2.0+	
47H	71	Получить текущий каталог	2.0+	
48H	72	Выделить блок памяти	2.0+	
49H	73	Освободить блок памяти	2.0+	
4AH	74	Изменить размер блока памяти	2.0+	
4BH	75	Выполнить программу (EXEC)	2.0+	
4CH	76	Завершить процесс с кодом возврата	2.0+	
4DH	77	Получить код возврата	2.0+	
4EH	78	Найти первый файл	2.0+	H
4FH	79	Найти следующий файл	2.0+	H
50H	80	Зарезервирована		
51H	81	Зарезервирована		
52H	82	Зарезервирована		
53H	83	Зарезервирована		
54H	84	Получить флаг проверки	2.0+	
55H	85	Зарезервирована		
56H	86	Переименовать файл	2.0+	
57H	87	Получить или установить дату и время создания файла	2.0+	H
58H	88	Получить или установить стратегию выделения	3.0+	
59H	89	Получить расширенную информацию об ошибке		
5AH	90	Создать временный файл	3.0+	H
5BH	91	Создать новый файл	3.0+	H
5CH	92	Запереть или отпереть район файла	3.0+	H
5DH	93	Зарезервирована		
5EH	94	Получить имя машины, получить или задать установку принтера	3.1+	
5FH	95	Перенаправить устройство	3.1+	
60H	96	Зарезервирована		
61H	97	Зарезервирована		
62H	98	Получить адрес PSP	3.0+	
63H	99	Получить таблицу лидирующих байтов для DBCS	2.25 (только)	
64H	100	Зарезервирована		
65H	101	Получить расширенную информацию о национальном формате	3.3+	
66H	102	Получить или установить кодовую страницу		
67H	103	Установить число дескрипторов	3.3+	
68H	104	Сбросить в файл	3.3+	H
69H	105	Зарезервирована		
6AH	106	Зарезервирована		
6BH	107	Зарезервирована		
6CH	108	Расширенное открытие файла	4.0+	H

[1] Файловая операция выполняется с помощью функций типа FCB или типа дескриптора.

Список функций прерывания Int 21H по группам

Hex	Dec	Описание	Версия	F/H
Символьный ввод-вывод				
01H	1	Ввод символа с эхо	1.0+	
02H	2	Вывод символа	1.0+	
03H	3	Дополнительный ввод	1.0+	
04H	4	Дополнительный вывод	1.0+	
05H	5	Вывод на принтер	1.0+	
06H	6	Непосредственный ввод-вывод через консоль	1.0+	
07H	7	Нефильтрованный ввод без эхо	1.0+	
08H	8	Ввод символа без эхо	1.0+	
09H	9	Вывести на экран строку	1.0+	
0AH	10	Буферизованный ввод с клавиатуры	1.0+	
0BH	11	Проверить состояние ввода	1.0+	
0CH	12	Сбросить буфер и затем ввести	1.0+	
Файловые операции				
0FH	15	Открыть файл	1.0+	F
10H	16	Закрыть файл	1.0+	F
11H	17	Найти первый файл	1.0+	F
12H	18	Найти следующий файл	1.0+	F
13H	19	Удалить файл	1.0+	F
16H	22	Создать файл	1.0+	F
17H	23	Переименовать файл	1.0+	F
23H	35	Получить размер файла	1.0+	F
29H	41	Выделить и перенести в FCB имя файла	1.0+	F
3CH	60	Создать файл	2.0+	H
3DH	61	Открыть файл	2.0+	H
3EH	62	Закрыть файл	2.0+	H
41H	65	Удалить файл	2.0+	H
43H	67	Получить или установить атрибуты файла	2.0+	
45H	69	Дублировать дескриптор	2.0+	
46H	70	Перенаправить дескриптор	2.0+	
4EH	78	Найти первый файл	2.0+	H
4FH	79	Найти следующий файл	2.0+	H
56H	86	Переименовать файл	2.0+	
57H	87	Получить или установить дату и время создания файла	2.0+	H
5AH	90	Создать временный файл	3.0+	H
5BH	91	Создать новый файл	3.0+	H
67H	103	Установить число дескрипторов	3.3+	
68H	104	Сбросить в файл	3.3+	H
6CH	108	Расширенное открытие файла	4.0+	H
Операции с записями				
14H	20	Последовательное чтение	1.0+	F
15H	21	Последовательная запись	1.0+	F
1AH	26	Установить адрес DTA	1.0+	
21H	33	Чтение в режиме прямого доступа	1.0+	F
22H	34	Запись в режиме прямого доступа	1.0+	F
24H	36	Установить номер относительной записи	1.0+	F
27H	39	Чтение блока в режиме прямого доступа	1.0+	F
28H	40	Запись блока в режиме прямого доступа	1.0+	F
2FH	47	Получить адрес DTA	2.0+	
3FH	63	Прочитать файл или устройство	2.0+	H
40H	64	Вывести запись в файл или на устройство	2.0+	H
42H	66	Установить указатель файла	2.0+	H
5CH	92	Запереть или отпереть район файла	3.0+	H

Продолжение таблицы

Hex	Dec	Описание	Версия	F/H
Операции с каталогом				
39H	57	Создать каталог	2.0+	
3AH	58	Удалить каталог	2.0+	
3BH	59	Установить текущий каталог	2.0+	
47H	71	Получить текущий каталог	2.0+	
Управлением диском				
0DH	13	Сбросить диск	1.0+	
0EH	14	Выбрать диск	1.0+	
19H	25	Получить текущий диск	1.0+	
1BH	27	Получить информацию о дисководе, действующем по умолчанию	1.0+	
1CH	28	Получить информацию о дисководе	2.0+	
2EH	46	Установить флаг проверки	1.0+	
36H	54	Получить информацию о выделении дисковода	2.0+	
54H	84	Получить флаг проверки	2.0+	
Управление процессом				
00H	0	Завершить процесс	1.0+	
26H	38	Создать новый PSP	1.0+	
31H	49	Завершить и оставить резидентной	2.0+	
4BH	75	Выполнить программу (EXEC)	2.0+	
4CH	76	Завершить процесс с кодом возврата	2.0+	
4DH	77	Получить код возврата	2.0+	
62H	98	Получить адрес PSP	3.0+	
Управление памятью				
48H	72	Выделить блок памяти	2.0+	
49H	73	Освободить блок памяти	2.0+	
4AH	74	Изменить размер блока памяти	2.0+	
58H	88	Получить или установить стратегию выделения	3.0+	
Сетевые функции				
5EH	94	Получить имя машины, получить или задать установку принтера	3.1+	
5FH	95	Перенаправить устройство	3.1+	
Время и дата				
2AH	42	Получить дату	1.0+	
2BH	43	Установить дату	1.0+	
2CH	44	Получить время	1.0+	
2DH	45	Установить время	1.0+	
Другие системные функции				
25H	37	Установить вектор прерываний	1.0+	
30H	48	Получить номер версии MS-DOS	2.0+	
33H	51	Получить или установить флаг BREAK, получить код загрузочного дисковода	2.0+	
35H	53	Получить вектор прерываний	2.0+	
38H	56	Получить или установить информацию о национальном формате	2.0+	

Продолжение таблицы

Hex	Dec	Описание	Версия	F/H
44H	68	Управление вводом-выводом (IOCTL)	2.0+	
59H	89	Получить расширенную информацию об ошибке	3.0+	
63H	99	Получить таблицу лидирующих байтов для DBCS	2.25 (только)	
65H	101	Получить расширенную информацию о национальном формате	3.3+	
66H	102	Получить или установить кодовую страницу	3.3+	
Зарезервированные функции				
18H	24	Зарезервирована		
1DH	29	То же		
1EH	30	«–»		
1FH	31	«–»		
20H	32	«–»		
32H	50	«–»		
34H	52	«–»		
37H	55	«–»		
50H	80	«–»		
51H	81	«–»		
52H	82	«–»		
53H	83	«–»		
55H	85	«–»		
5DH	93	«–»		
60H	96	«–»		
61H	97	«–»		
64H	100	«–»		
69H	105	«–»		
6AH	106	«–»		
6BH	107	«–»		

Int 20H
Завершить процесс

Завершает текущий процесс. Это один из нескольких способов, с помощью которого в программе можно осуществить окончательный выход. В этом случае MS-DOS выполняет следующие операции:

- Вся память, выделенная процессу, освобождается.
- Файловые буферы сбрасываются, а все дескрипторы файлов или устройств, открытые самим процессом, закрываются.
- Вектор обработчика завершения (прерывание Int 22H) восстанавливается по содержимому PSP:000AH.
- Вектор обработчика Ctrl-C (прерывание Int 23H) восстанавливается по содержимому PSP:000EH.
- [2:0+] Вектор обработчика критической ошибки (прерывание Int 24H) восстанавливается по содержимому PSP:0012H.
- Управление передается обработчику завершения.

Если программа возвращается в COMMAND.COM, то управление передается резидентной части последней, которая в случае необходимости перезагружается. Если выполняется командный файл, то выбирается и интерпретируется следующая строка

этого файла; в противном случае выдается запрос на следующую команду пользователя.

При вызове: CS = адрес сегмента префикса программы.
При возврате. Ничего не возвращается.
Замечания. Все файлы, записанные с помощью функций типа FCB, должны быть закрыты до осуществления выхода по этому обращению; в противном случае данные могут быть потеряны.

Другие способы осуществления окончательного завершения:
- с помощью прерывания Int 21H с функцией 00H
- с помощью прерывания Int 21H с функцией 31H
- с помощью прерывания Int 21H с функцией 4CH
- с помощью прерывания Int 27H

[2.0 +] Предпочтительнее осуществлять завершение с помощью прерывания Int 21H с функцией 31H или функцией 4CH, так как в этом случае можно передавать в родительский процесс код возврата.

[3.0 +] Если программа выполняется в сети, то до ее завершения необходимо освободить все запертые районы файлов.
Пример. Завершить текущую программу с передачей управления родительской программе:

```
.
.
.
int     20h             ; Переход в MS-DOS
```

Int 21H **[1.0]**
Функция 00H
Завершить процесс

Завершает текущий процесс. Это один из нескольких способов, с помощью которого в программе можно осуществить окончательный выход. В этом случае MS-DOS выполняет следующие операции:

- Вся память, выделенная процессу, освобождается.
- Файловые буферы сбрасываются, а все дескрипторы файлов или устройств, открытые самим процессом, закрываются.
- Вектор обработчика завершения (прерывание Int 22H) восстанавливается по содержимому PSP:000AH.
- Вектор обработчика Ctrl-C (прерывание Int 23H) восстанавливается по содержимому PSP:000EH.

[2.0+] Вектор обработчика критический ошибки (прерывание Int 24H) восстанавливается по содержимому PSP:0012H.

Управление передается обработчику завершения.

Если программа возвращается в COMMAND.COM, то управление передается резидентной части последней, которая в случае необходимости перезагружается. Если выполняется командный файл, то выбирается и интерпретируется следующая строка этого файла; в противном случае выдается запрос на следующую команду пользователя.

При вызове: AH = 00H
CS = адрес сегмента префикса программного сегмента
При возврате. Ничего не возвращается.
Замечания. Все файлы, записанные с помощью функций типа FCB, должны быть

закрыты до осуществления выхода по этому обращению; в противном случае данные могут быть потеряны.

Другие способы осуществления окончательного завершения:

- с помощью прерывания Int 21H с функцией 00H
- с помощью прерывания Int 21H с функцией 31H
- с помощью прерывания Int 21H с функцией 4CH
- с помощью прерывания Int 27H

[2.0+] Предпочтительнее осуществлять завершение с помощью прерывания Int 21H с функцией 31H или функцией 4CH, так как в этом случае можно передавать в родительский процесс код возврата.

[3.0+] Если программа выполняется в сети, то до ее завершения необходимо освободить все запертые районы файлов.

Пример. Завершить текущую программу с передачей управления родительской программе:

```
.
.
.
mov     ah,0            ; Номер функции
int     20h             ; Переход в MS-DOS
```

Int 21H
Функция 01H
Ввод символа с эхо

[1] Вводит символ с клавиатуры, а затем отображает его на экране дисплея. Если символ не готов, то ждет его появления.

[2.0+] Читает символ из устройства стандартного вывода. Если ожидающего символа нет, то ждет его появления. Ввод может перенаправляться. (Если было выполнено перенаправление ввода, то обнаружить признак EOF нельзя.)

При вызове. AH = 01H

При возврате. AL = 8-бит входные данные.

Замечания. Если стандартный ввод не перенаправляется и считывается комбинация клавиш Ctrl-C, то выполняется прерывание Int 23H. Если стандартный ввод перенаправляется и на консоли обнаруживается Ctrl-C, а обработка BREAK включена (BREAK ON), выполняется прерывание Int 23H.

Для чтения расширенных ASCII-кодов (например, кодов специальных функциональных клавиш F1–F10) на клавиатурах IBM PC и совместимых с ними компьютеров необходимо обращаться к данной функции дважды. При первом обращении возвращается значение 00H, что указывает на наличие расширенного кода.

Кроме того, см. прерывание Int 21H с функциями 06H, 07H и 08H, с помощью которых осуществляется ввод символа с разными комбинациями режима эхо и/или чувствительности к Ctrl-C.

[2.0+] Кроме того, можно осуществлять ввод с клавиатуры с помощью операции чтения (прерывание Int 21H с функцией 3FH), с использованием предопределенного дескриптора стандартного ввода (0000H) при условии, что ввод не перенаправлялся, или дескриптора, полученного при операции открытия логического устройства CON.

Пример. Осуществить ввод одного симовла с клавиатуры в регистр AL, отобразить его на экране дисплея (операция эхо) и сохранить его в переменной char:

```
char    db      0               ; Входной символ
        .
        .
        .
```

```
mov     ah,1            ; Номер функции
int     21h             ; Переход в MS-DOS
mov     char,al         ; Сохранить символ
.
.
.
```

Int 21H [1.0]
Функция 02H
Вывод символа

[1] Выводит символ на активный видеоэкран.

[2.0+] Выводит символ на устройство стандартного вывода. Вывод может перенаправляться. (В случае перенаправления вывода установить переполнение диска невозможно.)

При вызове. AH = 02H
DL = 8-бит данные для вывода

При возврате. Ничего не возвращается.

Замечания. Если после вывода затребованного символа обнаруживается нажатие на клавиатуре клавиш Ctrl-C, то выполняется прерывание Int 23H.

Если стандартный вывод не перенаправлялся, то код клавиши возврата на один символ заставляет курсор переместиться на одну позицию влево. Если стандартный вывод перенаправлялся, код клавиши возврата на один символ не обрабатывается каким-либо специальным образом.

[2.0+] Символьные строки можно пересылать на экран дисплея, осуществляя операцию записи (с помощью прерывания Int 21H с функцией 40H) с использованием предопределенного дескриптора стандартного вывода (0001H), при условии, что вывод не перенаправлялся, или дескриптора, полученного при открытии логического устройства CON.

Пример. Переслать на устройство стандартного вывода символ «*»:

```
.
.
.
mov     ah,2            ; Номер функции
mov     dl,'*'          ; Выходной символ
int     21h             ; Переход в MS-DOS
.
.
.
```

Int 21H [1.0]
Функция 03H
Дополнительный ввод

[1] Читает символ из первого последовательного порта.

[2.0+] Читает символ из стандартного дополнительного устройства. По умолчанию – это первый последовательный порт (COM1).

При вызове: AH = 03H

При возврате. AL – 8-бит входные данные.

Замечания. Для большинства версий системы MS-DOS последовательное устройство не обладает буферизацией и не управляется прерываниями. В том случае, когда дополнительное устройство посылает данные быстрее, чем их может обработать программа, символы могут быть потеряны.

В момент запуска на IBM PC система PC-DOS инициализирует первый последовательный порт следующим образом: 2400 бод без контроля четности, 1 стоп-бит, 8 бит данных. Другие реализации MS-DOS могут инициализировать последовательное устройство по-другому.

При обращении к данной функции в пользовательской программе невозможно прочитать состояние дополнительного устройства и обнаружить ошибки ввода-вывода (такие, как потеря символов). На компьютерах IBM PC наиболее строгий контроль можно получить с помощью прерывания Int 14H ПЗУ BIOS или путем прямого управления контроллером передачи данных.

В случае обнаружения нажатия на клавиатуре комбинации клавиш Ctrl-C выполняется прерывание Int 23H.

[2.0+] Кроме того, можно читать с дополнительного устройства, сделав запрос на операцию чтения (с помощью прерывания Int 23H с функцией 3FH) с использованием предопределенного дескриптора стандартного последовательного порта (0003H) или дескриптора, полученного в операции открытия логического устройства AUX.

Пример. Прочитать символ из стандартного дополнительного ввода и сохранить его в переменной char:

```
char    db      0               ; Входной символ
        .
        .
        .
        mov     ah,3            ; Номер функции
        int     21h             ; Переход в MS-DOS
        mov     char,al         ; Сохранить символ
        .
        .
        .
```

Int 21H **[1.0]**
Функция 04H
Дополнительный вывод

[1] Выводит символ в первый последовательный порт.

[2.0+] Выводит символ на стандартное дополнительное устройство. По умолчанию – это первый последовательный порт (COM1).

При вызове. AH = 04H
DL = 8-бит выходные данные

При возврате. Ничего не возвращается.

Замечания. Если устройство вывода занято, то данная функция ждет, пока устройство не будет готово принять символ.

С помощью данной функции невозможно опрашивать состояние дополнительного устройства. На машинах IBM PC более строгий контроль можно получать с помощью прерывания Int 14H ПЗУ BIOS или путем прямого управления контроллером передачи данных.

В случае обнаружения нажатия на клавиатуре комбинации клавиш Ctrl-C осуществляется прерывание Int 23H.

[2.0+] Кроме того, символьные строки можно пересылать на дополнительное устройство с помощью операции записи (прерывание Int 21H с функцией 40H) с использованием предопределенного дескриптора стандартного дополнительного устройства (0003H) и дескриптора, полученного при выполнении операции открытия логического устройства AUX.

Пример. Вывести на дополнительное устройство символ «*»:

```
.
.
.
mov     ah,4            ; Номер функции
mov     dl,'*'          ; Выходной символ
int     21h             ; Переход в MS-DOS
.
.
.
```

Int 21H **[1.0]**
Функция 05H
Вывод на принтер

[1] Пересылает символ на первое (логическое) устройство печати (PRN или LPT1).

[2.0+] Пересылает символ на стандартное устройство печати. По умолчанию таким устройством является принтер, связанный с первым параллельным портом (LPT1), если оно явным образом не перенаправлено пользователем с помощью команды MODE.

При вызове. AH = 05H
DL = 8-бит выходные данные.

При возврате. Ничего не возвращается.

Замечания.

Если принтер занят, то данная функция ждет, пока принтер не будет готов принять символ.

В среде MS-DOS не существует стандартного способа опросить состояние принтера.

В случае обнаружения нажатия на клавиатуре комбинации клавиш Ctrl-C осуществляется прерывание Int 23H.

[2.0+] Пересылать строки на принтер можно с помощью операции записи (прерывание Int 21H с функцией 40H) с использованием предопределенного дескриптора устройства стандартной печати или дескриптора, полученного при операции открытия логического устройства PRN или LPT1.

Пример. Вывести на устройство печати символ «*»:

```
.
.
.
mov     ah,5            ; Номер функции
mov     dl,'*'          ; Выходной символ
int     21h             ; Переход в MS-DOS
.
.
.
```

Int 21H **[1.0]**
Функция 06H
Непосредственный ввод-вывод через консоль

Используется программами, в которых необходимо выполнять операции чтения и записи всех допустимых символов (в том числе управляющих) без какого-либо вмешательства со стороны операционной системы.

[1] Выполняет операции чтения символа с клавиатуры и записи символа на экран.

[2.0+] Выполняет операции чтения символа с устройствами стандартного ввода и записи символа на устройство стандартного вывода. Операции ввода-вывода могут перенаправляться. (В случае перенаправления ввода-вывода невозможно обнаружить признак конца файла (EOF) и переполнение диска.)

При вызове. AH = 06H
DL = запрошенная функция
00H–FEH, если запрашивается вывод
0FFH, если запрашивается ввод

При возврате. Если при вызове DL = 00H–0FEH
Ничего не возвращается
Если при вызове DL = FFH и символ готов
Флаг нуля = сброшен
AL = 8-бит входные данные
Если при вызове DL = FFH и символ не готов
Флаг нуля = установлен

Замечания. При использовании данной функции в случае ввода Ctrl-C никаких специальных действий не выполняется.

На IBM PC и совместимых с ними компьютерах для чтения клавиш с расширенным ASCII-кодом (например, кодов специальных функциональных клавиш F1–F10) необходимо обращаться к данной функции дважды. При первом обращении возвращается значение 00H, что указывает на наличие расширенного кода.

См. также прерывание Int 21H с функциями 01H, 07H и 08H, которые обеспечивают символьный ввод с различными комбинациями операции эхо и/или реакцией на код Ctrl-C, и функциями 02H и 09H, которые можно применять для записи символов на стандартный вывод.

[2.0+] Можно выполнять ввод с клавиатуры, задавая операцию чтения (с помощью прерывания Int 21H с функцией 3FH) с предопределенным дескриптором стандартного ввода (0000H), при условии, что ввод не перенаправлялся, или дескриптором, полученным в операции открытия логического устройства CON.

[2.0+] Можно пересылать символы на экран, запрашивая операцию записи (с помощью прерывания Int 21H с функцией 40H) и предопределенным дескриптором стандартного вывода (0001H), при условии, что вывод не перенаправлялся, или дескриптором, полученным в операции открытия логического устройства CON.

Примеры. Переслать на устройство стандартного вывода символ «*»:

```
.
mov     ah,6            ; Номер функции
mov     dl,'*'          ; Выходной символ
int     21h             ; Переход в MS-DOS
.
.
.
```

Прочитать с устройства стандартного ввода символ и сохранить его в переменной char. Если ожидающего символа нет, ждать его появления:

```
char    db      0               ; Входной символ
        .
        .
        .
```

```
        mov     ah,6            ; Номер функции
        mov     dl,0ffh         ; Параметр - чтение
        int     21h             ; Переход в MS-DOS
        jz      wait            ; Ждать до готовности символа
        mov     char,al         ; Сохранить символ
        .
        .
        .
```

Int 21H **[1.0]**
Функция 07H
Нефильтрованный ввод без эхо

[1] Выполняет операцию чтения символа с клавиатуры без отображения его на экране дисплея. Если ожидающего символа нет, ждет его появления.

[2.0+] Выполняет операцию чтения символа с устройства стандартного ввода без отображения его на устройство стандартного вывода. Если ожидающего символа нет, ждет его появления. Ввод может перенаправляться. (В случае перенаправления ввода невозможно обнаружить признак конца файла EOF.)

При вызове. AH = 07H

При возврате. AL = 8-бит входные данные.

Замечания. При использовании данной функции в случае ввода Ctrl-C какого-либо специального действия не производится.

На IBM PC и совместимых с ними компьютерах для чтения клавиш с расширенным ASCII-кодом (например, кодов специальных функциональных клавиш F1–F10) требуется вызывать эту функцию дважды. При первом обращении возвращается значение 00H, что указывает на наличие расширенного кода.

См. также прерывание Int 21H с функциями 01H, 06H и 08H, которые обеспечивают ввод символа с различными комбинациями операции эхо и(или) реакцией на код Ctrl-C.

[2.0+] Можно осуществить ввод с клавиатуры, задавая операцию чтения (с помощью прерывания Int 21H с функцией 3FH) с предопределенным дескриптором стандартного ввода (0000H), при условии, что ввод не перенаправлялся, или дескриптором, полученным в операции открытия логического устройства CON.

Пример. Прочитать символ с устройства стандартного ввода без отображения его на экране дисплея и сохранить его в переменной char:

```
char    db      0               ; Входной символ
        .
        .
        .
        mov     ah,7            ; Номер функции
        int     21h             ; Переход в MS-DOS
        mov     char,al         ; Сохранить символ
        .
        .
        .
```

Int 21H **[1.0]**
Функция 08H
Ввод символа без эхо

[1] Выполняет операцию чтения символа с клавиатуры без его отображения на экране дисплея. Если ожидающего символа нет, ждет его появления.

[2.0+] Выполняет операцию чтения символа с устройства стандартного ввода без отображения его на устройство стандартного вывода. Если ожидающего символа нет, ждет его появления. Ввод может перенаправляться. (В случае перенаправления ввода невозможно обнаружить признак конца файла EOF).
При вызове. AH = 08H
При возврате. AL = 8-бит входные данные.
Замечания. Если перенаправление стандартного ввода не производилось и при операции чтения обнаруживается Ctrl-C, то осуществляется прерывание Int 23H. Если же производилось перенаправление стандартного ввода и при операции чтения с консоли обнаруживается Ctrl-C, то устанавливается флаг BREAK и осуществляется прерывание Int 23H. Если требуется избежать возможного прерывания по коду Ctrl-C, вместо этой функции используйте прерывание Int 21H с функцией 07H.

На IBM PC и совместимых с ними компьютерах для чтения клавиш с расширенным ASCII-кодом (например, кодов специальных функциональных клавиш F1 – F10) требуется вызывать эту функцию дважды. При первом обращении возвращается значение 00H, что указывает на наличие расширенного кода.

См. также прерывание Int 21H с функциями 01H, 06H и 07H, которые обеспечивают ввод символа с различными комбинациями операции эхо и(или) реакцией на код Ctrl-C.

[2.0+] Можно осуществлять ввод с клавиатуры с помощью операции чтения (прерывание Int 21H с функцией 3FH) с предопределенным дескриптором стандартного ввода (0000H) при условии, что ввод не перенаправлялся, или дескриптором, полученным в операции открытия логического устройства CON.
Пример. Прочитать символ с устройства стандартного ввода без его отображения на экране дисплея, допуская возможность обнаружения кода Ctrl-C; сохранить символ в переменной char:

```
char    db      0
        .
        .
        .
        mov     ah,8            ; Номер функции
        int     21h             ; Переход в MS-DOS
        mov     char,al         ; Сохранить символ
        .
        .
        .
```

Int 21H [1.0]
Функция 09H
Вывести на экран строку

[1] Пересылает строку символов на экран дисплея.

[2.0+] Пересылает строку символов на устройство стандартного вывода. Вывод может перенаправляться. (Если вывод перенаправлялся, то обнаружить переполнение диска невозможно.)
При вызове. AH = 09H
DS:DX = сегмент:относительный_адрес строки
При возврате. Ничего не возвращается.
Замечания. Строка должна завершаться символом $ (24H), который не передается. Все другие ASCII-коды, в том числе и управляющие, могут содержаться в строке.

См. также прерывание Int 21H с функциями 02H и 06H для вывода одного символа на экран дисплея или устройства стандартного вывода.

В случае обнаружения нажатия на клавиатуре комбинации клавиш Ctrl-C осуществляется прерывание Int 23H.

[2.0+] Можно пересылать строки на экран дисплея, выполняя операцию записи (с помощью прерывания Int 21H с функцией 40H) с предопределенным дескриптором стандартного вывода (0001H), при условии, что вывод не перенаправлялся, или дескриптором, полученным в операции открытия логического устройства CON.

Пример. Переслать строку символов Hello World, за которыми следуют коды возврата каретки и перевода строки, на устройство стандартного вывода:

```
cr      equ     0dh
lf      equ     0ah

msg     db      'Hello World',cr,lf,'$'
        .
        .
        .
        mov     ah,9            ; Номер функции
        mov     dx,seg msg      ; Адрес строки
        mov     ds,dx
        mov     dx,offset msg
        int  21h                ; Переход в MS-DOS
        .
        .
        .
```

Int 21H [1.0]
Функция 0AH(10)
Буферизованный ввод с клавиатуры

[1] Осуществляет операцию чтения строки с клавиатуры и размещения ее в созданном пользователем буфере. Символы отображаются на экране дисплея.

[2.0+] Осуществляет операцию чтения строки байта, вплоть до и включая ASCII-код возврата каретки (0DH), с устройства стандартного ввода и размещения ее в созданном пользователем буфере. Символы отображаются на устройстве стандартного вывода. Ввод может перенаправляться. (Если ввод перенаправлялся, то невозможно обнаружить признак конца файла EOF.)

При вызове. AH = 0AH

DS:DX = сегмент:относительный_адрес буфера

При возврате. Ничего не возвращается (данные помещены в буфер).

Замечания. Буфер, используемый данной функцией, имеет следующий формат:

Байт	Описание
0	максимальное число вводимых символов, установленное программой
1	фактическое число прочитанных символов (без кода возврата каретки), установленное системой MS-DOS
2 +	строка, прочитанная с клавиатуры или стандартного ввода, заканчивающаяся кодом возврата каретки (0DH).

Если в буфер помещено на один символ меньше, чем это определено максимальным числом символов, то последующий ввод игнорируется и звучит сигнал до тех пор, пока не обнаруживается ввода символа возврата каретки.

Эта функция обладает способностью вводить и помещать в буфер клавиатуры

символы, не ожидая запроса со стороны программы, причем в этом случае выполняются все команды редактирования при работе с клавиатурой.

Если стандартный ввод не перенаправлялся и обнаруживается нажатие на консоли комбинации клавиш Ctrl-C, то выполняется прерывание Int 23H. Если же стандартный ввод перенаправлялся и обнаруживается нажатие на консоли комбинации клавиш Ctrl-C, то устанавливается флаг BREAK и осуществляется прерывание Int 23H.

См. прерывание Int 21H с функциями 01H, 06H, 07H и 08H для ввода с клавиатуры или устройства стандартного ввода одиночного символа.

[2.0+] Строки можно вводить с клавиатуры, осуществляя операцию чтения (с помощью прерывания Int 21H с функцией 3FH) с предопределенным дескриптором стандартного ввода (0000H), при условии, что ввод не перенаправлялся, или дескриптором, полученным в операции открытия логического устройства CON.

Пример. Прочитать с устройства стандартного ввода строку, максимальная длина которой 80 символов, и поместить ее в буфер buff:

```
buff    db      81              ; Максимальная длина ввода
        db      0               ; Фактическая длина ввода
        db      81 dup (0)      ; Фактически ввод размещается здесь
        .
        .
        .
        mov     ah,0ah          ; Номер функции
        mov     dx,seg buff     ; Адрес буфера ввода
        mov     dx,offset buff
        int     21h             ; Переход в MS-DOS
        .
        .
        .
```

Int 21H **[1.0]**
Функция 0BH(11)
Проверить состояние ввода

[1] Проверяет, имеется ли символ с клавиатуры.

[2.0+] Проверяет, имеется ли символ с устройства стандартного ввода. Ввод может перенаправляться.

При вызове. AH = 0BH

При возврате. AL=00H, если символ отсутствует
FFH, если имеется по крайней мере один символ

Замечания. [1] В случае обнаружения Ctrl-C выполняется прерывание Int 23H.

[2.0+] Если стандартный ввод не перенаправлялся и обнаруживается нажатие на консоли комбинации клавиш Ctrl-C, то осуществляется прерывание Int 23H. Если же стандартный ввод перенаправлялся и обнаруживается нажатие на консоли комбинации клавиш Ctrl-C, то устанавливается флаг BREAK и выполняется прерывание Int 23H.

Если символ ждет, то данная функция будет непрерывно возвращать флаг со значением «истина» до тех пор, пока этот символ не будет обработан с помощью прерывания Int 21H с функцией 01H, 06H, 07H, 08H, 0AH или 3FH.

Данная функция эквивалентна IOCTL-функции, осуществляемой по прерыванию Int 21H с функцией 44H и подфункцией 06H.

Пример. Проверить, имеется ли символ с устройства стандартного ввода:

```
        .
        .
        .
        mov     ah,0bh          ; Номер функции
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Есть ли ждущий символ ?
        jnz     ready           ; Перейти, если символ готов
        .
        .
        .
```

Int 21H **[1.0]**
Функция 0CH(12)
Сбросить буфер и затем ввести

[1] Очищает буфер клавиатуры, а затем активизирует одну из функций ввода с клавиатуры.

[2.0+] Очищает буфер стандартного ввода, а затем активизирует одну из функций символьного ввода. Ввод может перенаправляться.

При вызове. AH = 0CH
AL = номер функции ввода, которая будет активизирована после сброса буфера (допускается 01H, 06H, 07H, 08H и 0AH).
(если AL = 0AH)
DS:DX = сегмент:относительный_адрес буфера ввода

При возврате. (если при вызове AL = 01H, 06H, 07H или 08H)
AL = 8-бит входные данные
(если при вызове AL = 0AH)
Ничего не возвращается (данные помещены в буфер)

Замечание. Наличие данной функции дает возможность программе отменить свойство упреждающего ввода системы MS-DOS. Она не учитывает никаких символов, ожидающих во внутреннем системном буфере упреждающего ввода, заставляя заданную функцию ввода ждать именно того символа, который вводится после программного запроса (обычно ввод выполняется с клавиатуры).

Наличие или отсутствие во время выполнения данной функции проверки кода Ctrl-C зависит от номера функции, помещенного в регистр AL.

Любая функция, номер которой указан в AL, кроме функций 01H, 06H, 07H, 08H и 0AH, сбрасывает буфер ввода и возвращает управление вызывающей программе.

Пример. Очистить буфер клавиатуры, затем ждать ввода символа, отобразить введенный символ на экране и далее вернуть его в AL. Сохранить символ в переменной char:

```
char    db      0
        .
        .
        .
        mov     ah,0ch          ; Номер функции
        mov     al,1            ; Подфункция = ввод символа
        int     21h             ; Переход в MS-DOS
        mov     char,al         ; Сохранить символ
        .
        .
        .
```

Int 21H [1.0]
Функция 0DH(13)
Сбросить диск

Сбрасывает все файловые буферы. Все данные, записанные логически пользовательскими программами и временно буферизованные в MS-DOS, записываются физически на диск.
При вызове. AH = 0DH
При возврате. Ничего не возвращается.
Замечания. Эта функция не обновляет записей в каталоге диска для файлов, которые остаются открытыми. Если ваша программа не закрыла соответствующим образом все файлы до удаления диска, а размеры файлов изменились, то данные, переносимые на диск с помощью данной функции, могут оказаться недоступными, так как записи в каталоге будут неправильными.

[3.3+] Предпочтительнее использовать вместо данной функции прерывание Int 21H с функцией 68H (Сбросить в файл), которое обеспечивает также и обновление каталога диска.
Пример. Сбросить все внутренние дисковые буферы системы MS-DOS:

```
.
.
.
mov     ah,0dh          ; Номер функции
int     21h             ; Переход в MS-DOS
.
.
.
```

Int 21H [1.0]
Функция 0EH(14)
Выбрать диск

Выбирает в качестве текущего заданный или действующий по умолчанию дисковод, а также возвращает общее число логических дисководов в системе.
При вызове. AH = 0EH
DL = код дисковода (0 = A, 1 = B и т. д.)
При возврате. AL = число логических дисководов в системе.
Замечания. [1] Доступно 16 обозначений дисковода (0–0FH).

[2] Доступно 63 обозначения дисковода (0–3FH).

[3.0+] Доступно 26 обозначений дисковода (0–19H).

Для сохранения совместимости снизу вверх в новых прикладных программах следует ограничиваться для обозначения дисковода буквами A–Z (0 = A, 1 = B и т. д.).

Под логическим дисководом понимаются все блочно-ориентированные устройства: гибкие диски, псевдодиски (электронные диски) и дисководы жестких дисков. Часто дисковод единственного физического диска разбивается на два или более логических дисковода.

[1] [2] В системах, совместимых с IBM PC с одним дисководом, в регистр AL возвращается значение 2, так как PC-DOS на одном физическом дисководе гибкого диска поддерживает два логических дисковода (A: и B:). Фактическое число физических дисководов в системе можно определить с помощью прерывания Int 11H ПЗУ BIOS.

[3.0+] В регистр AL возвращается либо значение 5, либо код дисковода, соответствующий записи LASTDRIVE в файле CONFIG.SYS (если она вообще имеется), в зависимости от того, какое значение больше.
Пример. Сделать дисковод B текущим (по умолчанию). Сохранить общее число логических дисководов системы в переменной drives:

```
drives  db      0
        .
        .
        .
        mov     ah,0eh          ; Номер функции
        mov     dl,1            ; Дисковод 1 = B
        int     21h             ; Переход в MS-DOS
        mov     drives,al       ; Сохранить полное число дисководов
        .
        .
        .
```

Int 21H **[1.0]**
Функция 0FH (15)
Открыть файл

Открывает файл и делает его доступным для последующих операций чтения-записи.

При вызове. AH = 0FH
DS:DX = сегмент:относительный_адрес блока управления файлами

При возврате. В случае успешного выполнения функции (файл найден)
AL = 00H
и FCB заполнен системой MS-DOS следующим образом:
поле дисковода (относительный адрес 00H) = 1 для дисковода A, 2 для дисковода B и т. д.
поле текущего блока (относительный адрес 0CH) = 00H
поле размера записи (относительный адрес 0EH) = 0080H
[2.0+] поле размера (относительный адрес 10H) = размер файла в каталоге
[2.0+] поле даты (относительный адрес 14H) = отметка даты в каталоге
[2.0+] поле времени (относительный адрес 16H) = отметка времени в каталоге
В случае невыполнения функции (файл не найден)
AL = 0FFH

Замечания. Если в программе вы намерены использовать размер записи, отличный от 128 байт, то после того, как файл будет успешно открыт, но до того, как будет выполнена любая другая дисковая операция, необходимо установить в FCB по относительному адресу 0EH поле размера записи.

Если должен быть осуществлен режим прямого доступа, то вызывающая программа после успешного открытия файла должна также установить поле относительной записи (относительный адрес 21H) FCB.

Для установки времени и даты создания каталога см. прерывание Int 21H с функцией 57H.

[2.0+] Предпочтительнее вместо данной функции использовать прерывание Int 21H с функцией 3DH, которое обеспечивает полный доступ к иерархической структуре каталога.

[3.0+] Если программа выполняется в сети, то для совместимости операций чтения-записи файл открывается в режиме разделения.

Пример. Попытаться открыть файл QUACK.DAT на действующем по умолчанию дисководе:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'QUACK   '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
        mov     ah,0fh          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если операция открытия
                                ; не выполнена
```

Int 21H **[1.0]**
Функция 10H(16)
Закрыть файл

Закрывает файл, сбрасывает внутренние системные буферы диска, связывающие данный файл с диском, а также обновляет каталог диска, если файл изменялся или расширялся.

При вызове. AH = 10H
DS:DX = сегмент:относительный_адрес блока управления файлами

При возврате. Если функция успешно выполнена (каталог успешно обновлен)
AL = 00H
Если функция не выполнена (файл в каталоге не найден)
AL = FFH

Замечание. [1] [2] Система MS-DOS версий 1 и 2 ненадежно обнаруживает смену гибкого диска, и возможна ошибка, если пользователь поменяет гибкие диски в тот момент, когда на дисководе имеется открытый файл. В наихудшем случае каталог и таблица размещения файлов вновь установленного диска могут быть повреждены или уничтожены.

[2.0+] Предпочтительнее вместо данной функции использовать прерывание Int 21H с функцией 3EH.

Пример. Используя блок управления файлами myfcb, закрыть ранее открытый файл:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'QUACK   '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
        mov     ah,10h          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
```

```
        jnz     error           ; Перейти, если операция закрытия
                                ; не выполнена
        .
        .
        .
```

Int 21H **[1.0]**
Функция 11H(17)
Найти первый файл

Ищет файл с соответствующим именем в текущем каталоге указанного диска.

При вызове. AX = 11H

DS:DX = сегмент: относительный_адрес блока управления файлами

При возврате. Если функция успешно выполнена (файл с соответствующим именем найден)

AL = 00H

и буфер по текущему адресу области пересылки диска (DTA) заполнен как неоткрытый обычный FCB или расширенный FCB, в зависимости от указанной функции типа FCB

Если функция не выполнена (файл с соответствующим именем не найден)

AL = FFH

Замечания. В целях установки в качестве DTA буфера адекватного размера до обращения к этой функции используйте прерывание Int 21H с функцией 1AH.

Во всех версиях MS-DOS допускается использование в имени файла шаблона групповых операций «?». В версиях 3.0 и старше в имени файла можно также использовать шаблон групповых операций «*». В случае использования шаблона групповых операций «?» или «*» данная функция возвращает первый файл с именем, соответствующим шаблону.

Для поиска файлов с атрибутами системный, скрытый только для чтения, а также каталога и метки тома должен применяться расширенный FCB.

Если применяется расширенный FCB, то его байт атрибута определяет тип выполняемого поиска. Если байт атрибута содержит 00H, то ищутся только обычные файлы. Если установлен бит атрибута, соответствующий метке тома, то будут возвращаться только метки тома (если они существуют). В случае установки любого другого значения атрибута или комбинации значений (таких, как скрытый, системный и только для чтения) поиск будет производиться как среди файлов с указанными атрибутами, так и среди файлов без специальных атрибутов.

[2.0+] Предпочтительнее вместо данной функции использовать прерывание Int 21H с функцией 4EH, которое обеспечивает полный доступ к иерархической структуре каталога.

Пример. Найти в текущем каталоге первый файл с расширением .COM:

```
buff    db      37 dup(0)       ; Получает результат поиска

myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      '????????'      ; Шаблон имени файла
        db      'COM'           ; Расширение = COM
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
```

```
                        ; Установка адреса DTA
mov     ah,1ah          ; Номер функции
mov     dx,seg buff     ; Адрес буфера
mov     ds,dx
mov     dx,offset buff
int     21h             ; Переход в MS-DOS

                        ; Поиск первого соответствия
mov     ah,11h          ; Номер функции
mov     dx,seg myfcb    ; Адрес FCB
mov     ds,dx
mov     dx,offset myfcb
int     21h             ; Переход в MS-DOS
or      al,al           ; Проверить состояние
jnz     error           ; Перейти, если соответствие не найдено
.
.
.
```

Int 21H **[1.0]**
Функция 12H(18)
Найти следующий файл

Если предыдущее обращение к прерыванию Int 21H с функцией 11H было успешным, то возвращает следующий файл с соответствующим именем (если такой существует).

При вызове. AH = 12H

DS:DX = сегмент : относительный_адрес блока управления файлами

При возврате. Если функция успешно выполнена (файл с соответствующим именем найден

AL = 00H

и буфер по текущему адресу области пересылки диска (DTA) устанавливается как неоткрытый обычный FCB или расширенный FCB, в зависимости от типа FCB, первоначально указанного при обращении к прерыванию Int 21H с функцией 11H

Если функция не выполнена (файлов с соответствующим именем больше не найдено)

AL = FFH

Замечания. Эта функция предполагает, что используемый в качестве входа FCB соответствующим образом инициализирован предварительным вызовом прерывания Int 21H с функцией 11 (и, возможно, последующим вызовом прерывания Int 21H с функцией 12H) и что в имени или расширении заданного для поиска файла содержится по крайней мере один символ шаблона групповых операций.

Как и в случае прерывания Int 21H с функцией 11H, важно до вызова данной функции использовать прерывание Int 21H с функцией 1AH с целью установить в качестве DTA буфер адекватного размера.

[2.0+] Предпочтительнее использовать вместо этой функции прерывание Int 21H с функциями 4EH и 4FH, которое обеспечивает полный доступ к иерархической структуре каталога.

Пример. Предполагая успешным предыдущее обращение к функции 11H, найти в текущем каталоге следующий файл с расширением .COM. Если с момента предыдущего поиска DTA не изменялась, то еще одного вызова функции 1AH не требуется:

```
buff    db      37 dup(0)       ; Получает результат поиска

my_fcb  db      0               ; Дисковод = действующий по умолчанию
        db      '????????'      ; Шаблон имени файла
        db      'COM'           ; Расширение = COM
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
                                ; Установка адреса DTA
        mov     ah,1ah          ; Номер функции
        mov     dx,seg buff     ; Адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        int     21h             ; Переход в MS-DOS

                                ; Поиск следующего соответствия
        mov     ah,12h          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если соответствие не найдено
        .
```

Int 21H **[1.0]**
Функция 13H(19)
Удалить файл

Удаляет все файлы с соответствующим именем, расположенные в текущем каталоге заданного или действующего по умолчанию диска.

При вызове. AH = 13H

DS:DX = сегмент:относительный адрес блока управления файлами

При возврате. Если функция успешно выполнена (файл или файлы удалены)

AL = 00H

Если функция не выполнена (соответствующие файлы не найдены, или по крайней мере один соответствующий файл имел атрибут только для чтения)

AL = FFH

Замечания. Допускается использование в имени файла шаблона групповых операций «?»; если в имени файла присутствует шаблон групповых операций «?» и имеется более одного файла с соответствующим именем, то все они будут удалены.

[2.0+] Предпочтительнее использовать вместо этой функции прерывание Int 21H с функцией 41H, которое обеспечивает полный доступ к иерархической структуре каталога.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами создания в каталоге, где находится удаляемый файл.

Пример. Удалить файл MYFILE.DAT из текущего каталога текущего диска:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'MYFILE  '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
```

```
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
        mov     ah,13h          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если операция удаления
                                ; не выполнена
```

Int 21H **[1.0]**
Функция 14H(20)
Последовательное чтение

Читает из файла следующий последовательный блок данных, а затем выполняет соответствующий инкремент указателя файла.

При вызове. AH = 14H
DS:DX = сегмент:относительный адрес ранее открытого блока управления файлами

При возврате. AL = 00H, если чтение успешно выполнено
01H, если конец файла
02H, если циклическое оборачивание в сегменте
03H, если в конце файла прочитана неполная запись

Замечания. Запись читается в память по текущему адресу области передачи диска (DTA), заданному последним обращением к прерыванию Int 21H с функцией 1AH. Если размер записи и положение буфера таковы, что может произойти либо переполнение в сегменте, либо циклический возврат, то функция не выполняется, возвращая код завершения 02H.

Число байтов данных, которые будут прочитаны, задается в поле размера записи (относительный адрес 0EH) блока управления файлами (FCB).

Положение файла данных, которые будут прочитаны, задается комбинацией значений поля текущего блока (относительный адрес 0CH) и поля текущей записи (относительный адрес 20H) блока управления файлами (FCB). Содержимое этих полей также автоматически увеличивается данной функцией.

Если в конце файла читается неполная запись, то она дополняется до затребованной длины нулями.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами доступа чтения в каталоге, содержащем читаемый файл.

Пример. Прочитать данные (1024 байт) из файла, заданного ранее открытым блоком управления файлами myfcb:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'QUACK   '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
```

```
        mov     ah,14h          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
                                ; Установка размера записи
        mov     word ptr myfcb+0eH,1024
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если операция чтения не выполнена
        .
        .
        .
```

Int 21H [1.0]
Функция 15H(21)
Последовательная запись

Записывает в файл следующий последовательный блок данных, а затем выполняет соответствующий инкремент указателя файла.

При вызове. AH = 15H
DS:DX = сегмент:относительный_адрес ранее открытого блока управления файлами

При возврате. Al = 00H, если запись успешно выполнена
01H, если диск переполнен
02H, если циклическое оборачивание в сегменте

Замечания. Запись выполняется (логически, не обязательно физически) на диск из памяти по текущему адресу области передачи диска (DTA), заданному последним вызовом прерывания Int 21H с функцией 1AH. Если размер записи и положение буфера таковы, что может произойти переполнение сегмента или циклический переход, то функция не выполняется, возвращая код завершения 02H.

Число байтов данных, которые будут записаны, задается в поле размера записи (относительный адрес 0EH) блока управления файлами (FCB).

Положение файла записываемых данных задается комбинацией значений поля текущего блока (относительный адрес 0CH) и поля текущей записи (относительный адрес 2CH) блока управления файлами (FCB). Содержимое этих полей также автоматически увеличивается данной функцией.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами доступа записи в каталоге, содержащем записываемый файл.

Пример. Записать данные (1024 байт) в файл, заданный ранее открытым блоком управления файлами myfcb:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'QUACK   '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
        mov     ah,15h          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
                                ; Установка размера записи
        mov     word ptr myfcb+0eH,1024
        int     21h             ; Переход в MS-DOS
```

```
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если операция записи не выполнена
        .
        .
        .
```

Int 21H **[1.0]**
Функция 16H(22)
Создать файл

Создает в текущем каталоге новую запись о файле, либо усекает уже существующий файл с тем же именем до нулевой длины. Открывает файл для последующих операций чтения-записи.

При вызове. AH = 16H
DS:DX = сегмент:относительный адрес неоткрытого блока управления файлами

При возврате. Если функция успешно выполнена (файл создан или усечен)
AL = 00H
и FCB заполнен системой MS-DOS следующим образом:
поле дисковода (относительный адрес 00H) = 1 для дисковода A,
2 для дисковода B и т. д.
поле текущего блока (относительный адрес 0CH) = 00H
поле размера записи (относительный адрес 0EH) = 0080H
[2.0+] поле размера (относительный адрес 10H) = размер файла в каталоге
[2.0+] поле даты (относительный адрес 14H) = отметка даты в каталоге
[2.0+] поле времени (относительный адрес 16H) = отметка времени в каталоге
Если функция не выполнена (каталог переполнен)
AL = FFH

Замечания. Данная функция должна применяться с осторожностью, так как существующий файл с заданным именем усекается до нулевой длины (т. е. все данные этого файла безвозвратно теряются).

Если данная функция вызывается с расширенным блоком управления файлами (FCB), то новому файлу при его создании можно придавать специальный атрибут, например, *скрытый* или *системный*, устанавливая соответствующий бит байта атрибутов расширенного FCB.

Так как данная функция также открывает файл, то последующий вызов прерывания Int 21H с функцией OFH не требуется.

Для установки времени и даты создания каталога см. прерывание Int 21H с функцией 57H.

[2.0+] Предпочтительнее вместо этой функции использовать прерывание Int 21H с функциями 3CH, 5AH, 5BH и 6CH, которое обеспечивает полный доступ к иерархической структуре каталога.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами создания в каталоге, в котором будет находиться новый файл.

Пример. Создать в текущем каталоге файл, используя для блока управления файлами имя myfcb:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'QUACK   '      ; Имя файла, 8 символов
```

```
db      'DAT'           ; Расширение, 3 символа
db      25 dup (0)      ; Остаток FCB
.
.
.
mov     ah,16h          ; Номер функции
mov     dx,seg myfcb    ; Адрес FCB
mov     ds,dx
mov     dx,offset myfcb
int     21h             ; Переход в MS-DOS
or      al,al           ; Проверить состояние
jnz     error           ; Перейти, если операция создания
                        ; не выполнена
```

Int 21H **[1.0]**
Функция 17H(23)
Переименовать файл

Изменяет имена всех соответствующих файлов из текущего каталога диска на заданном дисководе.

При вызове. AH = 17H
DS:DX = сегмент:относительный адрес «специального» блока управления файлами

При возврате. Если функция успешно выполнена (один или более файлов переименованы)
AL = 00H
Если функция не выполнена (нет соответствующих файлов или новое имя файла совпадает с именем уже существующего файла)
AL = FFH

Замечания. В специальном блоке управления файлами в прежних позициях (байты 0–0BH) содержится код дисковода, имя файла и расширение и, кроме того, второе имя файла, которое занимает 6 байт, следующие за первым именем (относительный адрес 11H).

В первом имени файла допускается использование шаблона групповых операций «?». Любой файл, соответствующий спецификации первого файла, будет переименован в соответствии со спецификацией второго файла.

Если спецификация второго файла содержит несколько шаблонов групповых операций «?», то соответствующие им буквы первого имени файла остаются без изменения.

Функция завершает свою работу, если новое назначаемое файлу имя совпадает с именем уже существующего файла.

[2.0+] Для переименования каталога можно использовать данную функцию с расширенным FCB.

[2.0+] Предпочтительнее вместо этой функции использовать прерывание Int 21H с функцией 56H, которое обеспечивает полный доступ к иерархической структуре каталога.

Пример. Изменить имя файла с OLDNAME.DAT на NEWNAME.DAT:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'OLDNAME '      ; Прежнее имя файла, 8 символов
```

```
        db      'DAT'           ; Прежнее расширение, 3 символа
        db      6 dup (0)       ; Зарезервированная область
        db      'NEWNAME '      ; Новое имя файла, 8 символов
        db      'DAT'           ; Новое расширение, 3 символа
        db      14 dup (0)      ; Зарезервированная область
        .
        .
        .
        mov     ah,17h          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если операция переименования
                                ; не выполнена
        .
        .
        .
```

Int 21H
Функция 18H(24)
Зарезервирована

Int 21H **[1.0]**
Функция 19H(25)
Получить текущий диск

Возвращает код текущего или действующего по умолчанию дисковода.
При вызове. AH = 19H
При возврате. AL = код дисковода (0 = 1, 1 = B и т.д.).
Замечания. Для установки дисковода, действующего по умолчанию, используйте прерывание Int 21H с функцией 0EH.

Некоторые другие функции прерывания Int 21H используют коды дисковода с 1 (т.е. 1 = A, 2 = B и т.д.) и сохраняют код 0 для дисковода, действующего по умолчанию.
Пример. Получить текущий дисковод и сохранить его код в переменной cdrive:

```
cdrive  db      0               ; Код текущего дисковода
        .
        .
        .
        mov     ah,19h          ; Номер функции
        int     21h             ; Переход в MS-DOS
        mov     cdrive,al       ; Сохранить код дисковода
        .
        .
        .
```

Int 21H **[1.0]**
Функция 1AH(26)
Установить адрес DTA

Задает адрес области пересылки диска (DTA), который будет использоваться при последующих обращениях к функциям типа FCB.

При вызове. AH = 1AH
DS:DX = сегмент:относительный_адрес области пересылки диска
При возврате. Ничего не возвращается.
Замечания. Если данная функция не вызывается программой, то DTA устанавливается по умолчанию в префиксе программного сегмента по относительному адресу 0080H (буфер длиной в 128 байт).

Вообще, программист должен следить за тем, чтобы задаваемая для использования область буфера была достаточно большой для любой дисковой операции. Исключением в этом является ситуация, когда MS-DOS обнаружит и аварийно завершит передачу на диск или с диска, приводящую к циклическому оборачиванию в сегменте.

Для определения адреса области передачи данных текущего диска можно использовать прерывание Int 21H с функцией 2FH.

Единственной операцией типа дескриптора, использующей адрес DTA, является поиск каталога (Int 21H с функциями 4EH и 4FH).
Пример. Установить область передачи диска по адресу буфера с меткой buff:

```
buff    db      128 dup(?)
        .
        .
        .
        mov     ah,1ah          ; Номер функции
        mov     dx,seg buff     ; Адрес области
        mov     ds,dx           ; пересылки диска
        mov     dx,offset buff
        int     21h             ; Переход в MS-DOS
        .
        .
        .
```

Int 21H [1.0]
Функция 1BH(27)
Получить информацию о дисководе, действующем по умолчанию

Получает выборочную информацию о дисководе действующего по умолчанию диска и указатель байта идентификации типа носителя в таблице размещения файлов.
При вызове. AH = 1BH
При возврате. Если функция выполнена успешно
AL = число секторов в кластере
DS:BX = сегмент:относительный_адрес байта идентификатора типа носителя
CX = размер физического сектора (байты)
DX = число кластеров действующего по умолчанию диска
Если функция не выполнена (неверный дисковод или критическая ошибка)
AL = FFH
Замечания. Байт идентификатора типа носителя имеет следующие значения:
0F0H – 3,5 дюйма, двухсторонний, 18 секторов или «другие»
0F8H – жесткий диск
0F9H – 5,25 дюйма, двухсторонний, 15 секторов или 3,5 дюйма, двухсторонний, 9 секторов
0FCH – 5,25 дюйма, односторонний, 9 секторов

0FDH – 5,25 дюйма, двухсторонний, 9 секторов
0FEH – 5,25 дюйма, односторонний, 8 секторов
0FFH – 5,25 дюйма, двухсторонний, 8 секторов

Для получения информации о дисках, отличающейся от той, что дается для действующего по умолчанию дисковода, используется прерывание Int 21H с функцией 1CH или 36H.

[1] Адрес, возвращаемый в DS:BX, указывает на копию первого сектора фактической таблицы размещения файлов FAT с идентификатором типа носителя в первом байте.

[2.0+] Адрес, возвращаемый в DS:BX, указывает только на копию байта идентификатора типа носителя из таблицы размещения файлов FAT диска; память, расположенная выше этого адреса, не может предназначаться для содержимого FAT или другой полезной информации. Если требуется прямое обращение к FAT, то используется прерывание Int 25H, которое обеспечивает чтение FAT в память.

Пример. Определить, на текущем дисководе установлен сменный или фиксированный диск:

```
.
.
.
mov     ah,1bh          ; Номер функции
int     21h             ; Переход в MS-DOS

                        ; Проверка байта идентификатора носителя
cmp     byte ptr [bx],0f8h
je      fixed           ; Перейти, если это жесткий диск
jmp     floppy          ; Иначе предполагаем,что это дискета
.
.
.
```

Int 21H **[2.0]**
Функция 1CH(28)
Получить информацию о дисководе

Получает информацию о характеристиках заданного диска и указателе байта идентификатора типа носителя из его таблицы распределения файлов.

При вызове. AH = 1CH
DL = код дисковода (0 = по умолчанию, 1 = A и т. д.)

При возврате. Если функция успешно выполнена
AL = число секторов в кластере
DS:BX = сегмент:относительный адрес байта идентификатора типа носителя
CX = размер физического сектора (байты)
DX = число кластеров действующего по умолчанию или заданного диска
Если функция не выполнена (неверный дисковод или критическая ошибка)
AL = FFH

Замечания. Байт идентификатора типа носителя имеет следующие значения:
0F0H – 3,5 дюйма, двухсторонний, 18 секторов или «другие»
0F8H – жесткий диск

0F9H – 5,25 дюйма, двухсторонний, 15 секторов или 3,5 дюйма, двухсторонний, 9 секторов
0FCH – 5,25 дюйма, односторонний, 9 секторов
0FDH – 5,25 дюйма, двухсторонний, 9 секторов
0FEH – 5,25 дюйма, односторонний, 8 секторов
0FFH – 5,25 дюйма, двухсторонний, 8 секторов

Вообще данное обращение идентично прерыванию Int 21H с функцией 1BH, за исключением возможности указать определенный дисковод. Кроме того, см. прерывание Int 21H с функцией 36H, которое возвращает аналогичную информацию.

[1] Адрес, возвращаемый в DS:BX, указывает на копию первого сектора фактической таблицы размещения файлов (FAT) с идентификатором типа носителя в первом байте.

[2.0+] Адрес, возвращаемый в DS:BX, указывает только на копию байта идентификатора типа носителя из таблицы размещения файлов (FAT) диска; память, расположенная над этим адресом, не может предназначаться для содержимого FAT или другой полезной информации. Если требуется прямое обращение к FAT, то используется прерывание Int 25H, которое обеспечивает чтение FAT в память.

Пример. Определить, на дисководе C установлен сменный или фиксированный диск:

```
.
.
.
mov     ah,1ch          ; Номер функции
mov     dl,3            ; Код дисковода 3 = C
int     21h             ; Переход в MS-DOS

                        ; Проверка байта идентификатора носителя
cmp     byte ptr ds:[bx],0f8h
je      fixed           ; Перейти, если это жесткий диск
jmp     floppy          ; Иначе предполагаем,что это дискета
.
.
.
```

Int 21H
Функция 1DH (29)
Зарезервирована

Int 21H
Функция 1EH(30)
Зарезервирована

Int 21H
Функция 1FH (31)
Зарезервирована

Int 21H
Функция 20H(32)
Зарезервирована

Int 21H **[1.0]**
Функция 21H(33)
Чтение в режиме прямого доступа

Читает выбранную запись из файла в память

При вызове. AH = 21H
DS:DX = сегмент:относительный_адрес ранее открытого блока управления файлами

При возврате. Al = 00H, если чтение успешно выполнено
01H, если конец файла
02H, если циклическое оборачивание в сегменте
03H, если в конце файла прочитана неполная запись

Замечания. Запись читается в память по текущему адресу области передачи диска (DTA), заданному последним обращением к прерыванию Int 21H с функцией 1AH. Программист должен следить за тем, чтобы эта область была достаточно большой для любой пересылаемой записи. Если размер и положение буфера таковы, что может произойти переполнение сегмента или циклический переход, то функция не выполняется, возвращая код завершения 02H.

Положение в файле читаемых данных определяется комбинацией значений поля относительной записи (относительный адрес 21H) и поля размера записи (относительный адрес 0EH) FCB. Размер записи по умолчанию – 128 байт.

В качестве побочного эффекта данной функции поле текущего блока (относительный адрес 0CH) и поле текущей записи (относительный адрес 20H) обновляются в соответствии с полем относительной записи.

Содержимое поля относительной записи FCB автоматически данной функцией не увеличивается; необходимо позаботиться в прикладной программе о соответствующем обновлении FCB, если требуется читать последовательные записи. Сравните с прерыванием Int 21H с функцией 27H, которое обеспечивает чтение нескольких записей за один вызов функции, а также автоматическое увеличение содержимого поля относительной записи.

Если в конце файла читается неполная запись, то она дополняется до затребованной длины записи нулями.

[3.0+] Если программа выполняется в сети, пользователь должен обладать правами доступа чтения в каталоге, содержащем читаемый файл.

Пример. Открыть файл MYFILE.DAT, установить длину записи 1024 байт, а затем прочитать запись № 4 из файла в буфер buff:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'MYFILE  '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB

buff    db      1024 dup (?)    ; Получает читаемые данные
        .
        .
        .
                                ; Операция открытия файла
        mov     ah,0fh          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить статус операции открытия
        jnz     error           ; Перейти, если файла нет

                                ; Установка адреса DTA
        mov     ah,1ah          ; Номер функции
        mov     dx,offset buff  ; Прочитать адрес буфера
```

```
int     21h             ; Переход в MS-DOS

                        ; Установка размера записи
mov     word ptr myfcb+0eh,1024

                        ; Установка номера записи
mov     word ptr myfcb+21h,4
mov     word ptr myfcb+23h,0

                        ; Операция чтения-записи
mov     ah,21h          ; Номер функции
mov     dx,offset myfcb ; Адрес FCB
int     21h             ; Переход в MS-DOS
or      al,al           ; Проверить состояние
jnz     error           ; Перейти, если операция чтения не выполнена
.
.
.
```

Int 21H **[1.0]**
Функция 22H(34)
Запись в режиме прямого доступа

Записывает данные из памяти в выбранную запись в файле.

При вызове. AH = 22H
DS:DX = сегмент:относительный_адрес ранее открытого блока управления файлами

При возврате. AL = 00H, если запись успешно выполнена
01H, если диск переполнен
02H, если циклическое оборачивание в сегменте, запись прервана

Замечания. Запись выполняется (логически, не обязательно физически) в файл из памяти по текущему адресу передачи диска, значение которого задано последним обращением к прерыванию Int 21H с функцией 1AH. Если размер и положение буфера таковы, что может произойти переполнение или циклический переход, то функция не выполняется, возвращая код завершения 02H.

Положение в файле данных, которые будут записаны, определяется комбинацией значений поля относительной записи (относительный адрес 21H) и поля размера записи (относительный адрес 0EH) FCB. Размер записи по умолчанию – 128 байт.

В качестве побочного эффекта данной функции поле текущего блока (относительный адрес 0CH) и поле текущей записи (относительный адрес 20H) обновляются в соответствии с полем относительной записи.

Содержимое поля относительной записи FCB автоматически данной функцией не увеличивается; в прикладной программе необходимо позаботиться о соответствующем обновлении FCB, если требуется прочитать последовательные записи. Сравните с прерыванием Int 21H с функцией 28H, которое дает возможность за одно обращение к функции выполнить много записей, а также автоматически увеличить содержимое поля относительной записи.

Если запись выводится за текущим концом файла, то пространство между старым концом файла и новой записью выделяется, но не инициализируется.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами доступа записи в каталог, содержащий файл, в который будет произведена запись.

Пример. Открыть файл MYFILE.DAT, установить длину записи 1024 байт, выполнить вывод записи № 4 в файл из буфера buff, а затем файл закрыть:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'MYFILE  '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB

buff    db      1024 dup (?)    ; Буфер для записи
        .
        .
        .
                                ; Операция открытия файла
        mov     ah,0fh          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить статус операции открытия
        jnz     error           ; Перейти, если файла нет

                                ; Установка адреса DTA
        mov     dx,offset buff  ; Адрес буфера
        mov     ah,1ah          ; Номер функции
        int     21h             ; Переход в MS-DOS

                                ; Установка размера записи
        mov     word ptr myfcb+0eh,1024

                                ; Установка номера записи
        mov     word ptr myfcb+21h,4
        mov     word ptr myfcb+23h,0

                                ; Выполнение операции вывода записи
        mov     ah,22h          ; Номер функции
        mov     dx,offset myfcb ; Адрес FCB
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если операция вывода записи
                                ; не выполнена

                                ; Операция закрытия файла
        mov     ah,10h          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если операция закрытия
                                ; не выполнена
```

Int 21H **[1.0]**
Функция 23H(35)
Получить размер файла

Ищет в текущем каталоге соответствующий файл; если он найден, то обновляет в FCB размер файла, задаваемый в виде числа записей.

При вызове. AH = 23H
DS:DX = сегмент:относительный_адрес неоткрытого блока управления файлами

При возврате. Если функция успешно выполнена (соответствующий файл найден)
AL = 00H
и в поле относительной записи
(относительный адрес 21H) FCB установлено количество записей в файле, округленное, если надо, до следующей полной записи
Если функция не выполнена (соответствующий файл не найден)
AL = FFH

Замечания. Соответствующее значение должно помещаться в поле размера записи (относительный адрес 0EH) FCB до вызова данной функции. Для этой функции размер записи по умолчанию не устанавливается. Сравните с функциями типа FCB открытия и создания (с помощью прерывания Int 21H с функциями 0FH и 16H), которые инициализируют FCB с размером записи по умолчанию в 128 байт.

В целях нахождения размера файла в байтах в поле размера записи нужно установить 1.

Так как нумерация записей начинается с нуля, то данную функцию можно использовать для установки указателя файла типа FCB в конец файла.

Пример. Определить размер файла MYFILE.DAT в байтах и вывести результат в регистры DX:AX:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'MYFILE  '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
        mov     ah,23h          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
                                ; Размер записи = 1 байт
        mov     word ptr myfcb+0eh,1
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если нет файла

                                ; Получение размера файла в байтах
        mov     word ptr myfcb+21h
        mov     word ptr myfcb+23h
        .
        .
        .
```

Int 21H **[1.0]**
Функция 24H(36)
Установить номер относительной записи

Устанавливает поле номера относительной записи блока управления файлами (FCB) в соответствии с положением в текущем файле, как это записано в открытом FCB.

При вызове. AH = 24H
DS:DX = сегмент:относительный_адрес ранее открытого блока управления файлами

При возврате. AL уничтожается (на другие регистры не влияет)

Поле относительной записи (относительный адрес 21H)
FCB обновлено

Замечания. Эта функция используется при переходе от последовательных операций ввода-вывода файла к операциям прямого доступа к файлу. Содержимое поля относительной записи (относительный адрес 21H) извлекается из полей размера записи (относительный адрес 0EH), текущего блока (относительный адрес 0CH) и текущей записи (относительный адрес 20H) блока управления файлами.

До вызова данной функции все четыре байта поля относительной записи (относительный адрес 21H) FCB должны быть установлены в нуль.

Пример. После серии последовательных передач записей, выполненных с помощью блока управления файлами myfcb, получить положение в файле текущей относительной записи и вывести номер записи в DX:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'MYFILE  '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
        mov     dx,seg myfcb    ; Сделаем FCB адресуемым
        mov     ds,dx

                                ; Установка значения поля относительной
                                ; записи в нуль
        mov     word ptr myfcb +21h,0
        mov     word ptr myfcb +23h,0

                                ; Установка номера записи
        mov     ah,24h          ; Номер функции
        mov     dx,offset myfcb ; Адрес FCB
        int     21h             ; Переход в MS-DOS

                                ; Загрузка номера записи в DX
        mov     dx,word ptr myfcb+21h
        .
        .
        .
```

Int 21H **[1.0]**
Функция 25H(37)
Установить вектор прерываний

Инициализирует вектор прерываний, настраивая его на программу обработки прерывания.

При вызове. AH = 25H,
Al = номер прерывания
DS:DX = сегмент:относительный_адрес программы обработки прерывания

При возврате. Ничего не возвращается.

Замечания. В корректных прикладных программах следует отдавать предпочтение данной функции, а не прямому редактированию таблицы векторов прерывания.

Прежде чем изменять содержимое вектора прерывания, следует с помощью прерывания Int 21H с функцией 35H получить и сохранить его исходное значение, чтобы это значение можно было восстановить до завершения работы программы.

Пример. Установить новый обработчик прерываний zdiv, предназначенный для исключительных ситуаций ЦП типа «деление на нуль»:

```
        .
        .
        .
        mov     ah,25h          ; Номер функции
        mov     al,0            ; Номер прерывания
        mov     dx,seg zdiv     ; Адрес обработчика прерывания
        mov     ds,dx
        mov     dx,offset zdiv
        int     21h             ; Переход в MS-DOS
        .
        .
        .
zdiv:                           ; Обработчик прерывания int 00h
        iret                    ; (ничего не делает )
```

Int 21H **[1.0]**
Функция 26H(38)
Создать новый PSP

Копирует префикс программного сегмента (PSP) выполняющейся в настоящее время программы по заданному адресу сегмента в свободной памяти, после чего обновляет новый PSP так, чтобы им могла воспользоваться другая программа.
При вызове. AH = 26H
DX = сегмент нового префикса программного сегмента
При возврате. Ничего не возвращается.
Замечания. После выполнения PSP программы копируется в новый сегмент, в новом PSP соответствующим образом обновляется информация о размере памяти, а начиная с относительного адреса 0AH, сохраняются текущие значения векторов завершения (прерывание Int 22H) обработчика Ctrl-C (прерывание Int 23H) и обработчика критической ошибки (прерывание Int 24H).

Данная функция не загружает другую программу и сама по себе не вызывает выполнение программы.

[2.0+] Предпочтительнее вместо этой функции использовать прерывание Int 21H с функцией 4BH (EXEC), которое может служить для загрузки и выполнения программ и оверлеев в формате .COM или .EXE.
Пример. Создать новый префикс программного сегмента на 64 Кбайт выше выполняющейся в настоящее время программы. В этом примере предполагается, что выполняющаяся программа была загружена как файл типа .COM, так что регистр CS указывает на ее PSP в течение всего времени выполнения программы. Если же выполняющаяся программа была загружена как файл типа .EXE, то адрес PSP необходимо получить с помощью прерывания Int 21H с функцией 62H (в среде MS-DOS версий 3.0 и старше) или путем сохранения первоначального содержимого регистров DS или ES при входе в программу:

```
        .
        .
        .
        mov     ah,26h          ; Номер функции
        mov     dx,cs           ; Сегмент PSP данной
```

```
                        ; программы
add     dx,1000h        ; Добавить 64 Кбайт в целях
                        ; получения адреса нового PSP
int     21h             ; Переход в MS-DOS
.
.
.
```

Int 21H **[1.0]**
Функция 27H(39)
Чтение блока в режиме прямого доступа

Читает из файла в память одну или несколько последовательных записей, начиная с указанного места файла.

При вызове. AH = 27H
CX = число записей при чтении
DS:DX = сегмент:относительный_адрес ранее открытого блока управления файлами

При возврате. AL = 00H, если прочитаны все затребованные записи
01H, если конец файла
02H, если циклическое оборачивание в сегменте
03H, если в конце файла прочитана неполная запись
CX = фактическое число прочитанных записей

Замечания. Записи читаются в память по текущему адресу области передачи диска, заданному последним обращением к прерыванию Int 21H с функцией 1AH. Программист должен следить за тем, чтобы эта область была достаточно большой для пересылаемой группы записей. Если размер и положение буфера таковы, что может произойти переполнение сегмента или циклический переход, то функция не выполняется, возвращая код завершения 02H.

Положение в файле читаемых данных определяется комбинацией значений поля относительной записи (относительный адрес 21H) и поля размера записи (относительный адрес 0EH) FCB. Размер записи по умолчанию – 128 байт.

После осуществления передачи содержимого диска содержимое полей FCB текущего блока (относительный адрес 0CH), текущей записи (относительный адрес 20H) и относительной записи (относительный адрес 21H) обновляется, указывая на следующую запись в файле.

Если в конце файла читается неполная запись, то она дополняется до затребованной длины нулями.

Сравните с прерыванием Int 21H с функцией 21H, которое за один вызов передает только одну запись и не обновляет содержимое поля относительной записи FCB.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами доступа чтения в каталоге, содержащем читаемый файл.

Пример. Прочитать четыре записи длиной 1024 байт каждая, начиная с записи № 8, в буфер buff; использовать при этом блок управления файлами myfcb:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'MYFILE  '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB

buff    db      4096 dup (?)    ; Буфер для данных
        .
        .
        .
```

```
                        ; Установка адреса DTA
mov     ah,1ah          ; Номер функции
mov     dx,seg buff     ; Адрес буфера
mov     ds,dx
mov     dx,offset buff
int     21h             ; Переход в MS-DOS

                        ; Установка номера относительной записи
mov     word ptr myfcb+21h,8
mov     word ptr myfcb+23h,0

                        ; Установка размера записи
mov     word ptr myfcb+0eh,1024

                        ; Чтение записей
mov     ah,27h          ; Номер функции
mov     cx,4            ; Число записей
mov     dx,offset myfcb ; Адрес FCB
int     21h             ; Переход в MS-DOS
or      al,al           ; Проверить состояние
jnz     error           ; Перейти, если операция чтения не выполнена
.
.
.
```

Int 21H **[1.0]**
Функция 28H(40)
Запись блока в режиме прямого доступа

Выводит одну или несколько последовательных записей из памяти в файл, начиная с указанного места файла.

При вызове. AH = 28H
CX = число выводимых записей
DS:DX = сегмент:относительный_адрес ранее открытого блока управления файлами

При возврате. AL = 00H, если выведены все затребованные записи
01H, если диск переполнен
02H, если циклическое оборачивание в сегменте
CX = фактическое число выполненных записей

Замечания. На диск записи выводятся (логически, не обязательно физически) из памяти по текущему адресу области передачи диска, значение которого задано последним обращением к прерыванию Int 21H с функцией 1AH. Если размер и положение буфера таковы, что может произойти переполнение сегмента или циклический переход, то функция не выполняется, возвращая код завершения 02H.

Положение в файле выводимых данных определяется комбинацией значений поля относительной записи (относительный адрес 21H) и поля размера записи (относительный адрес 0EH) FCB. Размер записи по умолчанию – 128 байт.

После осуществления передачи содержимого диска, содержимое полей FCB текущего блока (относительный адрес 0CH), текущей записи (относительный адрес 20H) и относительной записи (относительный адрес 21H) обновляется, указывая на следующую запись в файле.

Если данная функция вызывается при CX = 0, то на диск никакие данные не выводятся, но файл расширяется или обрезается до длины, задаваемой комбинацией значений полей FCB – размера записи (относительный адрес 0EH) и относительной записи (относительный адрес 21H).

Сравните с прерыванием Int 21H с функцией 22H, которое за один вызов передает только одну запись и не обновляет содержимое поля относительной записи FCB.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами доступа записи в каталоге, содержащем файл, в который будут выводиться записи.

Пример. На диск из буфера buff вывести четыре записи по 1024 байт каждая, начиная с записи № 8 и используя при этом блок управления файлами myfcb:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'MYFILE  '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB

buff    db      4096 dup (?)    ; Буфер для данных
        .
        .
        .
                                ; Установка адреса DTA
        mov     ah,1ah          ; Номер функции
        mov     dx,seg buff     ; Адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        int     21h             ; Переход в MS-DOS

                                ; Установка номера относительной записи
        mov     word ptr myfcb+21h,8
        mov     word ptr myfcb+23h,0

                                ; Установка размера записи
        mov     word ptr myfcb+0eh,1024

                                ; Вывод записей
        mov     ah,28h          ; Номер функции
        mov     cx,4            ; Число записей
        mov     dx,offset myfcb ; Адрес FCB
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если операция вывода записей
                                ; не выполнена
```

Int 21H **[1.0]**
Функция 29H(41)
Выделить и перенести в FCB имя файла

Выделяет текстовую строку и переносит ее в различные поля блока управления файлами (FCB).

При вызове. AH = 29H

AL = флаги управления выделением и переносом

Бит 3 = 1 в случае, когда поле расширения FCB будет модифицироваться, только если в строке, которая выделяется и переносится, задается расширение

= 0 в случае, когда поле расширения FCB будет модифицироваться в любом случае; если в строке, которая

выделяется и переносится, расширение отсутствует, то расширение FCB заполняется ASCII-кодами пробела

Бит 2 = 1 в случае, когда поле имени файла FCB будет модифицироваться, только если в строке, которая выделяется и переносится, задано имя файла

= 0 в случае, когда поле имени файла FCB будет модифицироваться в любом случае; если в строке, которая выделяется и переносится, имя файла отсутствует, то имя файла FCB заполняется ASCII-кодами пробела

Бит 1 = 1 в случае, когда байт идентификатора дисковода в FCB будет модифицироваться, только если в строке, которая выделяется и переносится, задан дисковод

= 0 в случае, когда байт идентификатора дисковода в FCB будет модифицироваться в любом случае; если в строке, которая выделяется и переносится, отсутствует спецификатор дисковода, то поле кода дисковода FCB заполняется (по умолчанию) 0

Бит 0 = 1, если ведущие разделители будут пропускаться (игнорироваться)

= 0, если ведущие разделители не будут пропускаться (игнорироваться)

DS:SI = сегмент:относительный_адрес строки

ES:DI = сегмент:относительный_адрес блока управления файлами

При возврате. AL = 00H, если нет шаблонов групповых операций

01H, если строка, которая выделена и переносится, содержит шаблоны групповых операций

FFH, если недопустимый спецификатор дисковода

DS:SI = сегмент:относительный_адрес первого символа после выделенного и переносимого имени файла

ES:DI = сегмент:относительный_адрес отформатированного неоткрытого блока управления файлами

Замечания. Данная функция рассматривает следующие символы как разделители:

[1] :.;, = + символ_табуляции / " [ ]

[2,0+] :.;, = + символ_табуляции

Данная функция рассматривает все разделители и приведенные ниже символы как признак завершения:

:.;, = + символ_табуляции < >| / " [ ]

Если в строке, которая будет выделена и перенесена, отсутствует допустимое имя файла, то после возврата ES:DI +1 указывает на ASCII-код пробела.

Если в имени файла или его расширении встречается символ шаблона «*», то этот и все последующие символы соответствующего поля FCB заменяются на символ «?».

Данную функцию (как и вообще блоки управления файлами) нельзя применять для работы с файлами, описание которых включает в себя путь.

Пример. Выделить строку fname и перенести ее в блок управления файлами myfcb:

```
fname   db      'D:QUACK.DAT',  ; Имя файла, предназначенное для выделения
                                ; и переноса

myfcb   db      37 dup (0)      ; Становится FCB
```

```
        .
        .
        .
        mov     ah,29h          ; Номер функции
        mov     al,01h          ; Пропустить начальные разделители
        mov     si,seg fname    ; Адрес имени файла
        mov     ds,si
        mov     si,offset fname
        mov     di,seg myfcb    ; Адрес FCB
        mov     es,di
        mov     di,offset myfcb
        int     21h             ; Переход в MS-DOS
        cmp     al,0ffh         ; Проверить состояние
        je      error           ; Перейти, если указан неверный дисковод
        .
        .
        .
```

Int 21H **[1.0]**
Функция 2AH(42)
Получить дату

Получает системные день месяца, день недели, месяц и год.
При вызове. AH = 2AH
При возврате. CX = год (1980–2099)
DH = месяц (1–12)
DL = день (1–31)
В среде MS-DOS версий 1.1 и старше
AL = день недели (0 = воскресенье, 1 = понедельник и т. д.)
Замечания. Формат регистра данной функции точно такой же, как это требуется в случае прерывания Int 21H с функцией 2BH (Установить дату).

Для нахождения дня недели любой даты можно использовать вместе данную функцию и прерывание Int 21H с функцией 2BH. Сначала с помощью функции 2AH получается и сохраняется текущая дата. Затем интересующая вас дата устанавливается с помощью функции 2BH, а затем через последующий вызов функции 2AH по данной дате определяется день недели. В заключение с помощью дополнительного обращения к функции 2BH, используя величины, полученные при первоначальном вызове функции 2AH, восстанавливается текущая дата.
Пример. Получить текущую дату и сохранить ее компоненты в переменных year, day и month:

```
year    dw      0
month   db      0
day     db      0
        .
        .
        .
        mov     ah,2ah          ; Номер функции
        int     21h             ; Переход в MS-DOS
        mov     year,cx         ; Сохранить год (слово)
        mov     month,dh        ; Сохранить месяц (байт)
        mov     day,dl          ; Сохранить день (байт)
        .
        .
        .
```

Int 21H **[1.0]**
Функция 2BH(43)
Установить дату

Устанавливает в драйвере системных часов заданную дату. Системное время не изменяется.
При вызове. AH = 2BH
CX = год (1980–2099)
DH = месяц (1–12)
DL = день (1–31)
При возврате. AL = 00H, если дата успешно установлена
FFH, если недопустимая дата (игнорируется)
Замечание. Формат регистра данной функции точно такой же, как это требуется для прерывания Int 21H с функцией 2AH (Получить дату).
Пример. Установить системную дату в соответствии с содержимым переменных year, day и month:

```
year    dw      0
month   db      0
day     db      0
        .
        .
        .
        mov     ah,2bh          ; Номер функции
        mov     cx,year         ; Получить год (слово)
        mov     dh,month        ; Получить месяц (байт)
        mov     dl,day          ; Получить день (байт)
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если указана неверная дата
        .
        .
        .
```

Int 21H **[(.0]**
Функция 2CH(44)
Получить время

Получает из драйвера системных часов реального времени время дня, переведенное в часы, минуты, секунды и сотые доли секунды.
При вызове. AH = 2CH
При возврате. CH = часы (0–23)
CL = минуты (0–59)
DH = секунды (0–59)
DL = сотые доли секунды (0–99)
Замечания. Формат регистра данной функции точно такой же, как это требуется для прерывания Int 21H с функцией 2DH (Установить время).

У большинства систем, совместимых с IBM PC, часы реального времени не обладают разрешением в сотые доли секунды. На таких машинах величина, возвращаемая в регистр DL с помощью данной функции, не является непрерывной.
Пример. Получить текущее время и сохранить две основные его компоненты в переменных hours и minutes:

```
hous    db      0
minutes db      0
```

```
        .
        .
        .
        mov     ah,2ch          ; Номер функции
        int     21h             ; Переход в MS-DOS
        mov     hours,ch        ; Сохранить часы (байт)
        mov     minutes,cl      ; Сохранить минуты (байт)
        .
        .
        .
```

Int 21H **[1.0]**
Функция 2DH (45)
Установить время

Устанавливает в системных часах реального времени заданные значения часов, минут, секунд и сотых долей секунды. Системная дата не изменяется.
При вызове. AH = 2DH
CH = часы (0–23)
CL = минуты (0–59)
DH = секунды (0–59)
DL = сотые доли секунд (0–59)
При возврате. AL = 00H, если время успешно установлено
FFH, если недопустимое время (игнорируется)
Замечание. Формат регистра данной функции точно такой же, как это требуется для прерывания Int 21H с функцией 2CH (Получить время).
Пример. Установить системное время в соответствии с содержимым переменных hours и minutes. Установить в нуль текущее значение секунд и сотых долей секунды:

```
hous    db      0
minutes db      0
        .
        .
        .
        mov     ah,2dh          ; Номер функции
        mov     cl,hours        ; Получить часы (байт)
        mov     cl,minutes      ; Получить минуты (байт)
        mov     dx,0            ; Установить целые секунды и сотые доли
                                ; секунды в нуль
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние
        jnz     error           ; Перейти, если время задано неверно
        .
        .
        .
```

Int 21H **[1.0]**
Функция 2EH(46)
Установить флаг проверки

Устанавливает и сбрасывает флаг операционной системы в целях автоматической проверки чтением после записи данных.
При вызове. AH = 2EH
AL = 00H, если флаг проверки сброшен
01H, если флаг проверки установлен
DL = 00H (MS-DOS версий 1 и 2)
При возврате. Ничего не возвращается.

Замечания. Так как проверка чтением после записи замедляет дисковые операции, то по умолчанию флаг проверки сбрасывается.

Если конкретный драйвер дискового устройства не поддерживает проверку чтением после записи, то данная функция не работает. С помощью прерывания Int 21H с функцией 54H можно определить текущее состояние флага проверки.

Кроме того, состояние флага проверки управляется с помощью команд системы MS-DOS VERIFY OFF и VERIFY ON.

Пример. Сохранить в переменной vflag текущее состояние системного флага проверки, а затем установить обязательную проверку всех последующих дисковых операций записи:

```
vflag   db      0               ; Предыдущее состояние флага проверки
        .
        .
        .
                                ; Получение состояния флага проверки
        mov     ah,54h          ; Номер функции
        int     21h             ; Переход в MS-DOS
        mov     vflag,al        ; Сохранить текущее состояние флага

        mov     ah,2eh          ; Номер функции
        mov     al,1            ; AL = 1 для включения операции проверк.:
        mov     dl,0            ; В регистре DL должен быть нуль
        int     21h             ; Переход в MS-DOS
        .
        .
        .
```

Int 21H **[2.0]**
Функция 2FH(47)
Получить адрес DTA

Получает текущий адрес области передачи диска (DTA) для операций чтения-записи типа FCB.

При вызове. AH = 2FH

При возврате. ES:BX = сегмент:относительный_адрес области передачи диска

Замечание. С помощью прерывания Int 21H с функцией 1AH устанавливается адрес области передачи диска. По умолчанию DTA – это буфер (128 байт) с относительным адресом 80H в префиксе программного сегмента.

Пример. Получить адрес области передачи текущего диска и сохранить его в переменной olddta:

```
olddta  dd      ?               ; Сохранить адрес области пересылки диска
                .
                .
                .
                mov     ah,2fh          ; Номер функции
                int     21h             ; Переход в MS-DOS

                                        ; Сохранение в виде указателя DWORD
                mov     word ptr olddta,bx
                mov     word ptr olddta+2,es
                .
                .
                .
```

Int 21H [2.0]
Функция 30H(48)
Получить номер версии MS-DOS

Возвращает номер версии установленной на компьютере операционной системы MS-DOS. Эта функция используется в прикладных программах для определения возможностей операционной среды.

При вызове. AH = 30H
AL = 00H

При возврате. Если выполняется в среде MS-DOS версии 1
AL = 00H
Если выполняется в среде MS-DOS версий 2.0 и старше
AL = старший номер версии (MS-DOS 3.10 = 3 и т. д.)
AH = младший номер версии (MS-DOS 3.10 = 0AH и т. д.)
BH = серийный номер фирмы-изготовителя (обычно зависит от фирмы-изготовителя; 00H для IBM PC-DOS, 0FFH и другие значения для MS-DOS)
BL:CX = серийный номер (24 бит) пользователя (необязательный, зависит от фирмы-изготовителя)

Замечания. Так как данная функция не определена в среде MS-DOS версии 1, ее всегда следует вызывать при AL = 00H. В среде MS-DOS версии 1 AL будет возвращаться неизменным.

В случае определения среды MS-DOS версии 1 необходимо следить за тем, чтобы выход не произошел неприемлемым образом. Например, в среде MS-DOS версии 1 недоступны прерывание Int 21H с функцией 4CH (Завершить процесс с кодом возврата), прерывание Int 21H с функцией 40H (Записать в файл или устройство), а также дескриптор стандартной ошибки. В таких случаях в программе следует вывести на экран с помощью прерывания Int 21H с функцией 09H сообщение об ошибке, а затем завершить ее работу с помощью прерывания Int 20H или прерывания Int 21H с функцией 00H.

Пример. Получить номер версии системы MS-DOS и, если работа происходит не в среде MS-DOS версий младше чем 2.0, завершить текущий процесс с выводом на экран сообщения об ошибке.

```
cr      equ     0dh             ; Код ASCII возврата каретки
lf      equ     0ah             ; Код ASCII перевода строки

msg     db      cr,lf
        db      'Неверная версия MS-DOS'
        db      cr,lf,'$'
        .
        .
        .
        mov     ax,3000h        ; Номер функции
        int     21h             ; Переход в MS-DOS
        cmp     al,2            ; Настоящая версия MS-DOS 2 или старше ?
        jae     label1          ; Да, перейти

                                ; Вывод на экран сообщения об ошибке
        mov     ah,09h          ; Номер функции
        mov     dx,offset msg   ; Адрес сообщения
        int     21h             ; Переход в MS-DOS
                                ; Завершение процесса
```

```
        mov     ah,0            ; Номер функции
        int     21h             ; Переход в MS-DOS

label1: .
        .
        .
```

Int 21H [2.0]
Функция 31H(49)
Завершить и оставить резидентной

Завершает выполнение текущей исполняемой программы, при этом передает родительскому процессу код возврата и резервирует часть или всю память, выделенную под программу, с целью не допустить ее перекрытия следующей загружаемой транзитной программой. Затем система MS-DOS выполняет следующие действия:

- Файловые буфера сбрасываются, а все дескрипторы файлов или устройств, открытые самим процессом, закрываются.
- Вектор обработчика завершения (прерывание Int 22H) восстанавливается по содержимому PSP:000AH.
- Вектор обработчика Ctrl-C (прерывание Int 23H) восстанавливается по содержимому PSP:000EH.

[2.0+] Вектор обработчика критической ошибки (прерывание Int 24H) восстанавливается по содержимому PSP:0012H.

Управление передается обработчику завершения.

В тех случаях, когда программа возвращается в COMMAND.COM, управление передается резидентной части последней, а транзитная ее часть, если это необходимо, перезагружается. В тех случаях, когда выполняется командный файл, выбирается и интерпретируется следующая строка файла; в противном случае выдается запрос на следующую команду пользователя.

При вызове. AH = 31H
AL = код возврата
DX = объем памяти (в параграфах) для резервирования

При возврате. Ничего не возвращается.

Замечания. Обычно вызов этой функции применяется для того, чтобы дать возможность написанным пользователям утилитам, драйверам и обработчикам прерывания загружаться как обычным программам типа .COM или .EXE и оставаться после этого резидентными. Последующий вход в программу осуществляется через аппаратное или системное прерывание.

Данная функция пытается установить начальный блок выделенной памяти длиной (в параграфах), заданной содержимым регистра DX. Если в прикладной программе через прерывание Int 21H с функцией 48H ранее были затребованы другие блоки памяти, то они с помощью данной функции не освободятся.

Другие способы выполнения окончательного выхода:

- с помощью прерывания Int 20H;
- с помощью прерывания Int 21H с функцией 00H;
- с помощью прерывания Int 21H с функцией 4CH;
- с помощью прерывания Int 27H.

Родительский процесс с помощью прерывания Int 21H с функцией 4DH (Получить код возврата) может извлечь код возврата. С помощью оператора IF ERROFLEVEL код возврата можно также проверить в командном файле. По соглашению значение

кода возврата, равное нулю, указывает на успешное выполнение, а отличное от нуля – на ошибку.

Данная функция не должна вызываться из программ типа .EXE, загружаемых в верхней части области транзитных программ (т. е. компонуемых с помощью ключа /HIGH), так как при этом резервируется память, которая обычно используется транзитной частью COMMAND. COM. Если программу COMMAND.COM нельзя перезагрузить, то система выйдет из строя.

[2.0+] Предпочтительнее вместо этой функции использовать прерывание Int 27H, которое поддерживает код возврата, позволяет резервировать больший объем памяти и не требует, чтобы в CS находился адрес сегмента префикса программного сегмента.

[3.0+] Если программа выполняется в сети, то до ее завершения необходимо освободить все ранее запертые районы файла.

Пример. Завершить программу с кодом возврата 1, но оставить ее резидентной, резервируя для процесса 16 Кбайт памяти начиная с префикса программного сегмента:

```
.
.
.
mov     ah,31h          ; Номер функции
mov     al,1            ; Возвратить код завершения
                        ; родительскому процессу
mov     dx,0400h        ; Число параграфов для резервирования
int     21h             ; Переход в MS-DOS
.
.
.
```

Int 21H
Функция 32H(50)
Зарезервирована

Int 21H **[2.0]**
Функция 33H(51)
Получить или установить флаг BREAK, получить код загрузочного дисковода

Получает или изменяет состояние флага BREAK операционной системы, которое влияет на обработку кода Ctrl-C при функциональных вызовах. Кроме того, в версии 4.0 возвращает код системного загрузочного дисковода.

При вызове. В случае получения флага BREAK
AH = 33H
AL = 00H
В случае установки флага BREAK
AH = 33H
AL = 01H
DL = 00H, если флаг BREAK сбрасывается
01H, если флаг BREAK устанавливается
[4] В случае получения кода загрузочного дисковода
AH = 33H
AL = 05H

При возврате. Если при вызове AL = 00H или 01H
DL = 00H флаг BREAK сброшен
= 01H флаг BREAK установлен

[4] Если при вызове AL = 05H
DL = код загрузочного дисковода (1 = A, 2 = B и т. д.)

Замечания. В тех случаях, когда системный флаг BREAK установлен, всякий раз, когда затребован системный ввод или вывод, осуществляется проверка на ввод с клавиатуры Ctrl-C; если Ctrl-C обнаруживается, то управление передается обработчику Ctrl-C (с помощью прерывания Int 23H). В тех случаях, когда флаг BREAK сброшен, система MS-DOS проверяет ввод Ctrl-C только при выполнении операций ввода-вывода с помощью традиционных символьно-ориентированных функций (прерывание Int 21H с функциями от 01H до 0CH).

Флаг BREAK не является только частью локальной среды текущей выполняемой программы; он влияет на все программы. Прикладные программы, которые изменяют состояние флага, в первую очередь должны сохранить исходное состояние флага, а затем перед своим завершением восстановить флаг.

Пример. Сохранить текущее состояние системного флага BREAK в переменной brkflag, а затем сбросить значение флага BREAK с целью блокировать проверку Ctrl-C во время обращения к большинству функций системы MS-DOS.

```
brkflag db      0               ; Сохранить флаг Break
        .
        .
        .
                                ; Получение текущего значения флага Break
        mov     ah,33h          ; Номер функции
        mov     al,0            ; AL = 0 для получения значения флага
        int     21h             ; Переход в MS-DOS
        mov     brkflag,dl      ; Сохранить текущее значение флага

                                ; Установка значения флага Break
        mov     ah,33h          ; Номер функции
        mov     al,1            ; AL = 1 для установки флага
        mov     dl,0            ; Сбросить флаг Break
        int     21h             ; Переход в MS-DOS
        .
        .
        .
```

Int 21H
Функция 344(52)
Зарезервирована

Int 21H **[2.0]**
Функция 35H(53)
Получить вектор прерываний

Получает адрес программы текущего обработчика прерываний заданного номера.

При вызове. AH = 35H
AL = номер прерывания

При возврате. ES:BX = сегмент:относительный_адрес обработчика прерываний

Замечание. Эта функция совместно с прерыванием Int 21H с функцией 25H (Установить вектор прерываний) применяется корректными прикладными программами с целью изменения или проверки таблицы векторов прерывания компьютера.

Пример. Получить адрес текущего обработчика прерывания для аппаратного уровня прерывания 0 (деление на нуль) и сохранить его в переменной oldint0:

```
oldint0 dd      ?               ; Адрес предыдущего обработчика прерываний
        .
        .
        .
        mov     ah,35h          ; Номер функции
        mov     al,0            ; Уровень прерывания
        int     21h             ; Переход в MS-DOS

                                ; Сохранение адреса старого обработчика
                                ; прерываний
        mov     word ptr oldint0,bx
        mov     word ptr oldint0+2,es
        .
        .
        .
```

Int 21H **[2.0]**
Функция 36H(54)
Получить информацию о характеристиках дисковода

Получает выборочную информацию о дисководе, из которой можно вычислить объем диска и оставшееся свободным пространство на диске.
При вызове. AH = 36H
DL = код дисковода (0 = по умолчанию, 1 = A и т. д.)
При возврате. Если функция выполнена успешно
AX = число секторов в кластере
BX = число доступных кластеров
CX = количество байтов в секторе
DX = число кластеров на дисководе
Если функция не выполнена (неверный дисковод)
AX = FFFFH
Замечания. Данная функция считает «потерянные» кластеры находящимися в использовании и при сообщении не включает их в число доступных кластеров, даже если они не назначены к файлу.

Аналогичная информация возвращается с помощью прерывания Int 21H с функциями 1BH и 1CH.
Пример. Вычислить объем (в байтах) диска на дисководе C, вывести результат в переменную drvsize. (Эта программа предполагает, что для представления произведения числа секторов в кластере на число байтов в секторе достаточно 16 бит.)

```
drvsize dd      ?               ; Объем диска C в байтах
        .
        .
        .
        mov     ah,36h          ; Номер функции
        mov     dl,3            ; Дисковод C = 3
        int     21h             ; Переход в MS-DOS

        mul     cx              ; Сектора/кластер
                                ; * байт/сектор
        mul     dx              ; * полное число кластеров
                                ;
                                ; Сохранение младшего слова
        mov     word ptr drvsize,ax
```

```
                        ; Сохранение старшего слова
        mov     word ptr drvsize+2,dx
        .
        .
        .
```

Int 21H
Функция 37H(55)
Зарезервирована

Int 21H **[2.0]**
Функция 38H(56)
Получить или установить информацию о национальном формате

[2] Получает информацию о национальном формате.

[3.0+] Получает информацию о национальном формате текущей или заданной страны, а также устанавливает код национального формата.

При вызове. При получении информации о национальном формате (MS-DOS версия 2)
AH = 38H
AL = 0 – для получения информации о национальном формате «текущей» страны
DS:DX = сегмент:относительный_адрес буфера для возвращаемой информации
При получении информации о национальном формате (MS-DOS версий 3.0 и старше)
AH = 38H
AL = 0 – для получения информации о национальном формате «текущей» страны
1 – FEH – для получения информации, если код национального формата < 255
FFH – для получения информации, если код национального формата ⩾ 255
BX = код страны, если AL = FFH
DS:DX = сегмент:относительный_адрес буфера для возвращаемой информации
При установке текущего кода национального формата (MS-DOS версий 3.0 и старше)
AH = 38H
AL = 1 – FEH, код национального формата, если его значение < 255
FFH, если значение кода ⩾255
BX = код национального формата, если A = 0FFH
DX = FFFFH

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
И при получении информации
BX = код национального формата
DS:DX = сегмент:относительный_адрес буфера, содержащего информацию
и буфер заполнен следующим образом:
(для PC-DOS 2.0 и 2.1)

Байт(ы)	Описание
00H–01H	формат даты 0-США месяц, день, год 1-Европа день, месяц, год 2-Япония год, месяц, день
02H–03H	код ASCIIZ идентификатора валюты
04H–05H	код ASCIIZ разделителя тысяч
06H–07H	код ASCIIZ десятичной точки
08H–1FH	зарезервированы

(для MS-DOS версий 2.0 и старше, PC-DOS версий 3.0 и старше)

Байт(ы)	Описание
00H–01H	формат даты 0-США месяц, день, год 1-Европа день, месяц, год 2-Япония год, месяц, день
02H–06H	код ASCIIZ строки идентификатора валюты
07H–08H	код ASCIIZ символа разделителя тысяч
09H–0AH	код ASCIIZ символа десятичной точки
0BH–0CH	код ASCIIZ символа разделителя даты
0DH–0EH	код ASCIIZ символа разделителя времени форматов валюты бит 0 = 0, если идентификатор валюты предшествует значению = 1, если идентификатор валюты следует за значением бит 1 = 0, если нет пробела между значением и идентификатором валюты = 1, если между значением и идентификатором валюты один пробел бит 2 = 0, если идентификатор валюты и десятичная точка разделены = 1, если идентификатор валюты заменяет десятичную точку
10H	число цифр после десятичной точки в представлении валюты
11H	формат времени бит 0 = 0, если 12-часовое представление = 1, если 24-часовое представление
12H–15H	адрес вызова процедуры отображения символов верхнего регистра
16H–17H	код ASCIIZ разделителя данных
18H–21H	зарезервированы

Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Действующий по умолчанию код национального формата определяется в файле CONFIG.SYS директивой COUNTRY или файлом драйвера клавиатуры KEYBxx, если таковой загружен. В противном случае код национального формата, действующий по умолчанию, определяется фирмой-изготовителем.

Предыдущее содержимое регистра CX может изменяться подфункцией «Получить информацию о национальном формате».

Процедура отображения символов верхнего регистра – это сегмент:относительный_адрес процедуры с атрибутом FAR, которая осуществляет отображение симво-

лов национального формата на значения символов от 80H до 0FFH. При вызове процедуры в регистре AL должен находиться отображаемый символ. Если для этого символа существует другое значение, то последнее возвращается в AL, в противном случае значение AL не изменяется. Обычно символы нижнего регистра отображаются в эквивалентные им символы верхнего регистра, а акцентированные или иным способом модифицированные гласные отображаются в эквивалентные им простые гласные.

[3.0+] MS-DOS использует содержимое регистра DX для различия подфункций «Получить» и «Установить информацию о национальном формате».

[3.3+] По прерыванию Int 21H с функцией 65H (Получить расширенную информацию о национальном формате) возвращается надмножество той информации, которую обеспечивает данная функция.

Примеры. Получить в буфер ctrybuf информацию о национальном формате:

```
ctrybuf db      34 dup (0)
        .
        .
        .
        mov     ah,38h          ; Номер функции
        mov     al,0            ; Получить код текущего национального формата
        mov     dx,seg ctrybuf  ; Адрес буфера для информации
        mov     ds,dx           ; о национальном формате
        mov     dx,offset ctrybuf
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        .
        .
        .
```

Если программа выполняется в среде PC-DOS версии 3.3 и текущий код национального формата 49 (Западная Германия), то буфер ctrybuf заполняется следующей информацией:

```
        dw      0001h           ; Формат даты
        db      'DM',0,0,0      ; Код ASCIIZ идентификатора валюты
        db      '.',0           ; Код ASCIIZ разделителя тысяч
        db      '.',0           ; Код ASCIIZ десятичной точки
        db      '.',0           ; Код ASCIIZ разделителя даты
        db      '.',0           ; Код ASCIIZ разделителя времени
        db      02h             ; Текущий формат
        db      02h             ; Число цифр после десятичной точки
        db      01h             ; Формат времени
        dd026ah:176ch           ; Адрес процедуры отображения символов
                                ; верхнего регистра
        db      ';',0           ; Код ASCIIZ разделителя таблицы данных
        db      10 dup (0)      ; Зарезервировано
```

Int 21H **[2.0]**
Функция 39H(57)
Создать каталог

Создает каталог по указанному пути на заданном дисководе.

При вызове. AH = 39H
DS:DX = сегмент:относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. Функция не выполняется, если:
– не существует хотя бы одного элемента пути;
– уже существует каталог с тем же именем в конце того же пути;
– родительский каталог нового каталога является корневым, и он переполнен;
– [3.0+] программа выполняется в сети, а ее пользователь имеет недостаточные права доступа.

Пример. На дисководе C в корневом каталоге создать новый каталог с именем MYSUB:

```
dname   db      'C:\MYSUB',0
        .
        .
        .
        mov     ah,39h          ; Номер функции
        mov     dx,seg dname    ; Адрес пути
        mov     ds,dx
        mov     dx,offset dname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция создания
                                ; не выполнена
```

Int 21H **[2.0]**
Функция 3AH(58)
Удалить каталог

Удаляет каталог на заданном дисководе по указанному пути.

При вызове. AH = 3AH
DS:DX = сегмент:относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. Функция не выполняется, если:
– не существует хотя бы одного элемента пути;
– заданный каталог является также текущим;
– заданный каталог содержит файлы;
– [3.0+] программа выполняется в сети, а ее пользователь имеет недостаточные права доступа.

Пример. На дисководе C в корневом каталоге удалить каталог с именем MYSUB:

```
dname   db      'C:\MYSUB',0
        .
        .
        .
        mov     ah,3ah          ; Номер функции
        mov     dx,seg dname    ; Адрес имени пути
        mov     ds,dx
```

```
mov     dx,offset dname
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если операция удаления
                        ; не выполнена
```

Int 21H [2.0]
Функция 3BH(59)
Установить текущий каталог

Устанавливает на заданном дисководе по указанному пути текущий или действующий по умолчанию каталог.

При вызове. AH = 3BH
DS:DX = сегмент:относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Функция не выполняется, если не существует какого-нибудь элемента пути.

С целью дальнейшего восстановления текущего каталога целесообразно перед вызовом прерывания Int 21H с функцией 47H, выбирающей другой каталог, использовать прерывание Int 21H с функцией 3BH для получения имени текущего каталога.

Пример. На дисководе C сменить текущий каталог на каталог\MYSUB:

```
dname   db      'C:\MYSUB',0
        .
        .
        .
    mov     ah,3bh          ; Номер функции
    mov     dx,seg dname    ; Адрес пути
    mov     ds,dx
    mov     dx,offset dname
    int     21h             ; Переход в MS-DOS
    jc      error           ; Перейти, если неправильный путь
    .
    .
    .
```

Int 21H [2.0]
Функция 3CH(60)
Создать файл

По заданному в коде ASCIIZ пути создает новый файл на указанном или действующем по умолчанию дисководе в указанном или действующем по умолчанию каталоге. Если указанный файл уже существует, он усекается до нулевой длины. В любом случае открывается файл и возвращается дескриптор, который может использоваться в программе для выполнения последующих операций доступа к файлам.

При вызове. AH = 3CH
CX = атрибут файла (можно комбинировать биты)

Бит(ы)	**Описание** (если бит установлен)
0	только для чтения
1	скрытый
2	системный
3	метка тома
4	зарезервирован (0)
5	атрибут архива
6–15	зарезервированы (0)

DS:DX = сегмент:относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
AX = дескриптор
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Функция не выполняется, если:

– не существует хотя бы одного элемента пути;

– файл создается в корневом каталоге, который уже полон;

– в заданном каталоге уже существует файл с тем же именем и атрибутом *только для чтения*;

– [3.0+] программа выполняется в сети, а ее пользователь имеет недостаточные права доступа.

При создании файла обычно ему не присваивается никаких специальных атрибутов (0). С помощью прерывания Int 21H с функцией 43H атрибуты файла можно последовательно изменять.

[3.0+] Метку тома можно создать с помощью атрибута 0008H, если она ранее не существовала. При создании файлов бит 3 параметра атрибутов всегда следует сбрасывать (0).

[3.0+] См. входные данные для прерывания Int 21H с функциями 5AH и 5BH, которые также можно использовать для создания файлов.

[4.0+] С помощью прерывания Int 21H с функцией 6CH можно объединить сервис, предоставляемый функциями 3CH, 3DH и 5BH.

Пример. Создать и открыть или обрезать до нулевой длины и открыть файл C:\MYDIR\MYFILE.DAT, а также сохранить для последующего доступа к файлу дескриптор:

```
fname   db      'C:\MYDIR\MYFILE.DAT',0

fhandle dw      ?
        .
        .
        .
        mov     ah,3ch          ; Номер функции
        xor     cx,cx           ; Без специальных атрибутов
        mov     dx,seg fname    ; Адрес пути
        mov     ds,dx
        mov     dx,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция создания
                                ; не выполнена
        mov     fhandle,ax      ; Сохранить дескриптор файла
        .
        .
        .
```

Int 21H **[2.0]**
Функция 3DH (61)
Открыть файл

По данному в коде ASCIIZ пути открывает заданный файл на указанном или действующем по умолчанию дисководе в указанном или действующем по умолчанию каталоге. Возвращает дескриптор, который может использоваться программой для последующего доступа к файлу.

При вызове. AH = 3DH
AL = режим доступа

Бит(ы)	Описание
0–2	режим доступа 000 = чтение 001 = запись 010 = чтение/запись
3	зарезервирован(0)
4–6	режим разделения (MS-DOS версий 3.0 и старше) 000 = режим совместимости 010 = ничего не допускается 010 = не допускается запись 011 = не допускается чтение 100 = разрешается все
7	флаг наследования (MS-DOS версий 3.0 и старше) 0 = дочерний процесс наследует дескриптор 1 = дочерний процесс не наследует дескриптор

DS:DX = сегмент:относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = дескриптор
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. С помощью данной функции будет открыт файл с соответствующим именем с любым атрибутом: *без специальных атрибутов, системный* или *скрытый.* Если файл имеет атрибут *только для чтения*, то успех операции открытия определяется также кодом доступа, находящегося в битах 0–2 регистра AL. После открытия файла указатель операций чтения-записи файла устанавливается по нулевому относительному адресу (первый байт файла).

Функция не выполняется, если:

– не существует хотя бы одного элемента пути;

– файл открывается с режимом доступа чтение-запись, но имеет атрибут *только для чтения;*

– [3.0+] Была загружена программа SHARE.EXE, и файл уже был открытым одним или несколькими другими процессами в режиме разделения, который несовместим с текущим программным запросом.

После успешного выполнения операции открытия с помощью прерывания Int 21H с функцией 57H можно получить доступ к записи о времени и дате создания файла.

С помощью прерывания Int 21H с функцией 43H можно получить атрибут файла (скрытый, системный, только для чтения, архивируемый).

В тех случаях, когда дескриптор файла наследуется дочерним процессом или

дублируется с помощью прерывания Int 27H с функцией 45H или 46H, то также наследуются и все ограничения на доступ и разделение файла.

[2] В регистре AL значимы только биты 0–2; остальные биты в целях совместимости снизу вверх должны быть обнулены.

[3.0+] Биты 4–7 регистра AL управляют доступом к файлу других программ. (Если программа SHARE.EXE. не загружена, то биты 4–6 не имеют значения.)

[3.0+] Ошибка разделения файла вызывает исключение критической ошибки (прерывание Int 24H) с кодом ошибки 02H. Для получения информации о нарушении режима разделения можно использовать прерывание Int 21H с функцией 59H.

[4.0+] С помощью прерывания Int 21H с функцией 6CH можно объединить сервис, представляемый функциями 3CH, 3DH и 5BH.

Пример. Открыть файл C:\MYDIR\MYFILE.DAT, предназначенный для операций чтения и записи, и сохранить дескриптор для последующего доступа к файлу:

```
fname   db      'C:\MYDIR\MYFILE.DAT',0

fhandle dw      ?
        .
        .
        .
        mov     ah,3dh          ; Номер функции
        xor     al,2            ; Режим = чтение - запись
        mov     dx,seg fname    ; Адрес пути
        mov     ds,dx
        mov     dx,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция открытия
                                ; не выполнена
        mov     fhandle,ax      ; Сохранить дескриптор файла
        .
```

Int 21H **[2.0]**
Функция 3EH(62)
Закрыть файл

По данному дескриптору, который получен при последней успешно выполненной операции открытия или создания, сбрасывает все внутренние буферы, связанные с файлом, выводя из них данные на диск, закрывает файл и освобождает для последующего использования дескриптор. Если файл изменялся, то в записи каталога о файле обновляются дата и время создания, а также размер файла.

При вызове. AH = 3EH
BX = дескриптор

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. Если вы случайно вызывете эту функцию с нулевым дескриптором, то устройство стандартного ввода будет закрыто, а клавиатура «замрет». Вы всегда должны вызывать функцию закрытия с правильным, ненулевым дескриптором.

Пример. Закрыть файл, дескриптор которого сохраняется в переменной fhandle:

```
fhandle dw      0
        .
        .
        .
        mov     ah,3eh          ; Номер функции
        mov     bx,fhandle      ; Дескриптор файла
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция закрытия
                                ; не выполнена
```

Int 21H [2.0]
Функция 3FH(63)
Прочитать файл или устройство

По данному допустимому дескриптору файла, полученному из последней операции открытия или создания, по адресу буфера и его длине (в байтах) передает данные в соответствии с текущим положением указателя файла из файла в буфер, а затем обновляет положение указателя файла.

При вызове. AH = 3FH
BX = дескриптор
CX = число байтов для чтения
DS:DX = сегмент:относительный_адрес буфера

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = число переданных байтов
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Если чтение с символьно-ориентированного устройства (например, стандартного ввода) происходит в режиме ASCII, то будет читаться максимально одна строка ввода (т.е. до символа возврата каретки или до достижения заданной длины в зависимости от того, что встретится раньше).

Если флаг переноса возвращается сброшенным, но AX = 0, то при запросе программы на чтение указатель файла уже был в конце файла.

Если флаг переноса возвращается сброшенным, но AX < CX, то в этом случае в конце файла была прочитана неполная запись или произошла ошибка.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами доступа чтения каталога и файла.

Пример. С помощью дескриптора, полученного из последней операции открытия или создания, прочитать 1024 байт в буфер buff, начиная с текущего положения указателя файла:

```
buff    db      1024 dup (?)    ; Буфер для чтения

fhandle dw      ?               ; Содержит дескриптор файла
        .
        .
        .
        mov     ah,3fh          ; Номер функции
        mov     dx,seg buff     ; Адрес буфера
        mov     ds,dx
```

```
        mov     dx,offset buff
        mov     bx,fhandle      ; Дескриптор файла
        mov     cx,1024         ; Число байт для чтения
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция чтения не выполнена

        cmp     ax,cx           ; Проверить число прочитанных байт
        jl      done            ; Перейти, если конец файла
        .
        .
        .
```

Int 21H **[2.0]**
Функция 40H(64)
Записать в файл или на устройство

При заданном допустимом дескрипторе файла, полученном из последней операции открытия или создания, адресе буфера и его длине (в байтах) передает данные по текущему положению указателя файла из буфера в файл, а затем обновляет положение указателя файла.

При вызове. AH = 40H
BX = дескриптор
CX = число байтов для вывода записи
DS:DX = сегмент:относительный_адрес буфера

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = число переданных байтов
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Если флаг переноса возвращается сброшенным, но AX < CX, то в этом случае в конце файла была прочитана неполная запись или произошла ошибка. Это могло быть обусловлено кодом Ctrl-Z (1AH), встроенным в данные, если адресат – символьно-ориентированное устройство в режиме ASCII, или переполнением диска, если адресат – файл.

Если при вызове функции CX = 0, то файл усекается или расширяется до текущего положения указателя файла.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами доступа записи в каталог и файл.

Пример. С помощью дескриптора, полученного из последней операции открытия или создания, вывести запись (1024 байт) из буфера buff на диск, начиная с текущего положения указателя файла:

```
buff    db      1024 dup (?)    ; Буфер для записи

fhandle dw      ?               ; Содержит дескриптор файла
        .
        .
        mov     ah,40h          ; Номер функции
        mov     dx,seg buff     ; Адрес буфера
        mov     ds,dx
        mov     dx,offset buff
```

```
        mov     bx,fhandle      ; Дескриптор файла
        mov     cx,1024         ; Длина для вывода записи
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция вывода записи
                                ; не выполнена
        cmp     ax,1024         ; Запись заполнена целиком ?
        jne     error           ; Нет, перейти
        .
        .
        .
```

Int 21H **[2.0]**
Функция 41H(65)
Удалить файл

Удаляет файл из указанного каталога с заданного или действующего по умолчанию диска.

При вызове. AH = 41H
DS:DX = сегмент:относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Эта функция удаляет файл тем, что замещает первый символ имени файла в каталоге символом σ(e5H) и отмечает кластеры, выделенные файлу, «свободными» в таблице размещения файлов диска. Фактические данные, находящиеся в таких кластерах, не «затираются». С помощью данной функции за одно обращение можно удалить только один файл. В отличие от функции «Удалить файл» типа FCB (с помощью прерывания Int 21H с функцией 13H) при использовании данной функции в спецификации файла не допускаются шаблоны групповых операций * и ?.

Функция не выполняется, если:

– отсутствует хотя бы один элемент пути;

– указанный файл существует, но имеет атрибут *только для чтения*. (Прежде чем попытаться удалить файл, можно использовать прерывание Int 21H с функцией 43H с целью проверить и изменить его атрибут.)

– [3.0+] программа выполняется в сети, а ее пользователь не имеет достаточных прав доступа.

Пример. На дисководе C в каталоге \MYDIR удалить файл с именем MYFILE.DAT:

```
fname   db      'C:\MYDIR\MYFILE.DAT',0
        .
        .
        .
        mov     ah,41h          ; Номер функции
        mov     dx,seg fname    ; Адрес имени файла
        mov     ds,dx
        mov     dx,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция удаления
                                ; не выполнена
```

Int 21N **[2.0]**
Функция 42H(66)
Установить указатель файла

Устанавливает указатель байта в файле относительно начала файла, конца файла или текущего положения в файле.

При вызове. AH = 42H
AL = код метода
00H абсолютное смещение от начала файла
01H знаковое смещение от текущей позиции указателя файла
02H знаковое смещение от конца файла
BX = дескриптор
CX = старшая половина смещения
DX = младшая половина смещения

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
DX = старшая половина результирующего указателя файла
AX = младшая половина результирующего указателя файла
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания: Данная функция для установки указателя файла использует код метода и величины с двойной точностью (32-бит представление). Следующая запись, которая читается и выводится в файл, будет начинаться с новой позиции указателя файла. Независимо от того, какой метод используется при обращении к этой функции, возвращаемый в DX:AX указатель файла – это всегда результирующее абсолютное смещение в байтах от начала файла.

Путем обращения к прерыванию Int 21H с функцией 42H при относительном адресе 0 и проверке возвращаемого положения указателя метод 02H можно использовать для нахождения размера файла.

С помощью методов 01H и 02H можно установить указатель файла в позицию до начала файла. При выполнении этого данная функция не возвращает ошибки. Но ошибка возникнет при последующей попытке прочитать или записать файл.

Примеры. С помощью дескриптора файла, полученного из последней операции открытия или создания, установить текущую позицию указателя файла через 1024 байт от начала файла:

```
fhandle dw      ?
        .
        .
        .
        mov     ah,42h          ; Номер функции
        mov     al,0            ; Метод = абсолютное значение
        mov     bx,fhandle      ; Дескриптор файла
        mov     cx,0            ; Старшая половина смещения
        mov     dx,1024         ; Младшая половина смещения
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        .
        .
        .
```

Приведенная ниже подпрограмма вводит номер записи, размер записи и дескриптор, а также устанавливает соответствующим образом указатель файла

```
; При вызове этой подпрограммы BX = дескриптор
;                               AX = номер записи
;                               CX = размер записи
; Подпрограмма возвращает все регистры в измененном состоянии
;
setptr  proc    near
        push    ax              ; Сохранить номер записи
        push    cx              ; Сохранить размер записи
        push    dx              ; Сохранить то, что находится в DX
        mul     cx              ; Получить произведение номера записи и
                                ; размера записи
        mov     cx,ax           ; Старшая часть произведения в CX,
        xchg    cx,dx           ; младшая часть - в DX
        mov     ax,4200h        ; Номер функции и метод
        int     21h             ; Переход в MS-DOS
        pop     dx              ; Восстановить предыдущее содержимое DX
        pop     cx              ; Восстановить размер записи
        pop     ax              ; Восстановить номер записи
        ret                     ; Вернуться к месту вызова
        setptr  endp
```

Int 21H **[2.0]**
Функция 43H(67)
Получить или установить атрибуты файла

Получает или изменяет атрибуты файла (только для чтения, скрытый, системный или архивируемый) и каталога.

При вызове. AH = 43H
AL = 00H получить атрибуты
01H установить атрибуты
CX = атрибут файла, если AL = 01H (биты могут комбинироваться)

Бит(ы)	**Описание** (если установлен)
0	только для чтения
1	скрытый
2	системный
3–4	зарезервирован(ы) (0)
5	атрибут архива
6–15	зарезервирован(ы) (0)

DS:DX = сегмент:относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
CX = атрибут файла

Бит(ы)	**Описание** (если установлен)
0	только для чтения
1	скрытый
2	системный
3	метка тома
4	каталог
5	атрибут архива
6–15	зарезервирован(ы) 0

Если функция не выполнена

Флаг переноса = установлен
AX = код ошибки

Замечания. При вызове этой функции биты 3 и 4 регистра CX всегда должны быть сброшены (0); другими словами, вы не можете изменить существующий в каталоге файл или метку тома. Однако с помощью этой функции вы можете присваивать атрибут *скрытый* существующему каталогу.

[3.0+] Если программа выполняется в сети, то пользователь должен обладать правами доступа создания в каталоге, где находится файл, атрибут которого должен быть изменен.

Пример. Изменить атрибут файла D:\MYDIR\MYFILE.DAT и сделать его доступным только для чтения с тем, чтоб другие прикладные программы не могли случайно его изменить или удалить:

```
rdonly  equ     01h             ; Атрибуты файла
hidden  equ     02h
system  equ     04h
volume  equ     08h
subdir  equ     10h
archive equ     20h

fname   db      'D:\MYDIR\MYFILE.DAT',0
        .
        .
        .
        mov     ah,43h          ; Номер функции
        mov     al,01h          ; Подфункция = модификация
        mov     al,rdonly       ; Атрибут файла = только для чтения
        mov     dx,seg fname    ; Адрес файла
        mov     ds,dx
        mov     dx,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция
                                ; модификации не выполнена
```

Int 21H **[2.0]**
Функция 44H(68)
Управление вводом-выводом (IOCTL)

Обеспечивает прямую связь между прикладной программой и драйвером устройства. Дает возможность программе получать аппаратно-зависимую информацию и запрашивать операции, которые не поддерживаются другими функциями MS-DOS.

Ниже приведен список подфункций IOCTL с номером версии MS-DOS, где они впервые стали доступными:

Подфункция	Описание	Версия MS-DOS
00H	Получить информацию о внешнем устройстве	2.0
01H	Установить информацию о внешнем устройстве	2.0
02H	Принять управляющие данные из драйвера символьно-ориентированного устройства	2.0

Подфункция	Описание	Версия MS-DOS
03H	Переслать управляющие данные в драйвер символьно-ориентированного устройства	2.0
04H	Принять управляющие данные из драйвера блочно-ориентированного устройства	2.0
05H	Переслать управляющие данные в драйвер блочно-ориентированного устройства	2.0
06H	Проверить состояние ввода	2.0
07H	Проверить состояние вывода	2.0
08H	Проверить, является ли блочно-ориентированное устройство сменным	3.0
09H	Проверить, является ли блочно-ориентированное устройство удаленным	3.1
0AH(10)	Проверить, является ли дескриптор удаленным	3.1
0BH(11)	Изменить счетчик повторных попыток разделения	3.1
0CH(12)	Родовое управление вводом-выводом символьно-ориентированных устройств	
	CL = 45H: Установить счетчик числа итераций	3.2
	CL = 4AH: Выбрать кодовую страницу	3.3
	CL = 4CH: Начать подготовку кодовой страницы	
	CL = 4DH: Закончить подготовку кодовой страницы	3.3
	CL = 5FH: Установить информацию о дисплее	4.0
	CL = 65H: Прочитать счетчик числа итераций	3.2
	CL = 6AH: Запросить выбранную кодовую страницу	3.3
	CL = 6BH: Запросить список подготовленных страниц	3.3
	CL = 7FH: Получить информацию о дисплее	4.0
0DH(13)	Родовое управление вводом-выводом блочно-ориентированных устройств	
	CL = 40H: Установить параметры внешнего устройства	3.2
	CL = 41H: Записать дорожку	3.2
	CL = 42H: Форматировать и проверить дорожку	4.0
	CL = 47H: Установить флаг доступа	4.0
	CL = 60H: Получить параметры внешнего устройства	3.2
	CL = 61H: Прочитать дорожку	3.2
	CL = 62H: Проверить дорожку	3.2
	CL = 67H: Получить флаг доступа	4.0
0EH(14)	Получить карту логического дисковода	3.2
0FH(15)	Установить карту логического дисковода	3.2

Для дескрипторов, связанных с файлами, можно применять только следующие подфункции IOCTL: 00H, 06H и 07H. Подфункции 00H–08H не поддерживаются для устройств, работающих в сети.

Int 21H **[2.0]**
Функция 44H(68) Подфункция 00H
IOCTL: Получить информацию о внешнем устройстве

Возвращает информационное слово устройства для файла или устройства, связанного с заданным дескриптором.

При вызове. AH = 44H
AL = 00H
BX = дескриптор
При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
DX = информационное слово устройства
Для файла:

Бит (ы)	Описание
0–5	номер дисковода (0 = A, 1 = B и т.д.)
6	0, если файл уже записан 1, если файл еще не записан
7	0, указатель файла
8–15	зарезервированы

Для устройства:

Бит (ы)	Описание
0	1, если стандартный ввод
1	1, если стандартный вывод
2	1, если NUL-устройство
3	1, если часы
4	зарезервирован
5	0, если дескриптор в режиме ASCII 1, если дескриптор в двоичном режиме
6	0, если на вводе конец файла
7	1, указатель устройства
8–13	зарезервированы
14	0, если подфункции IOCTL 02H и 03H не поддерживаются 1, если подфункции IOCTL 02H и 03H поддерживаются
15	зарезервирован

Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Биты 8–15 регистра DX соответствуют старшим 8 бит слова атрибута драйвера устройства.

Бит 5 слова информации устройства в отношении дескриптора, связанного с символьно-ориентированным устройством, показывает, рассматривает ли система MS-DOS дескриптор в двоичном режиме или режиме ASCII. В режиме ASCII система MS-DOS фильтрует символьный поток и при обнаружении комбинации символов Ctrl-C, Ctrl-S, Ctrl-P, Ctrl-Z, а также символа возврата каретки может выполнить специальные действия. В двоичном режиме все символы обрабатываются как данные и всегда читается или записывается точное число затребованных символов.

Пример. См. прерывание Int 21H с функцией 44H подфункцией 01H.

Int 21H **[2.0]**
Функция 44H(68) Подфункция 01H
IOCTL: Установить информацию о внешнем устройстве

Устанавливает определенные флаги для дескриптора, связанного с символьно-ориентированным устройством. Для дескриптора, связанного с файлом, эту подфункцию можно не использовать.

При вызове. AH = 44H
AL = 01H
BX = дескриптор
DX = слово информации устройства

Бит (ы)	Описание
0	1, если стандартный ввод
1	1, если стандартный вывод
2	1, если NUL-устройство
3	1, если часы
4	зарезервирован (0)
5	0 для выбора режима ASCII 0 для выбора двоичного режима
6	зарезервирован (0)
7	1, указатель устройства
8–15	зарезервированы (0)

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Если содержимое DH отличается от 00H, то управление передается программе, при этом флаг переноса установлен и в регистре AX – код ошибки 0001H (недопустимая функция).

Бит 5 слова информации устройства для дескриптора, связанного с символьно-ориентированным устройством, показывает, рассматривает ли система MS-DOS дескриптор в двоичном режиме или режиме ASCII. См. *Замечания* для прерывания Int 21H с функцией 44H(68), подфункцией 00H.

Пример. Установить дескриптор стандартного вывода в двоичный режим. Так как проверка после вывода каждого символа на коды Ctrl-C, Ctrl-S и Ctrl-P блокируется, то это повысит скорость вывода:

```
                        ; Получение информации об устройстве
mov     ax,4400h        ; Функция и подфункция
mov     bx,1            ; Дескриптор стандартного вывода
int     21h             ; Переход в MS-DOS

mov     dh,0            ; Установить DH = 0
or      dl,20h          ; Установить бит двоичного режима

                        ; Получение информации об устройстве
mov     ax,4401h        ; Функция и подфункция
int     21h             ; Переход в MS-DOS
.
.
.
```

Int 21H **[2.0]**

Функция 44H(68), подфункция 02H

IOCTL: Прочитать управляющие данные из драйвера символьно-ориентированного устройства

Читает управляющие данные из драйвера символьно-ориентированного устройства. Длина и содержание данных задаются для каждого драйвера без соблюдения какого-то стандартного формата. Эта функция необязательно приводит к какому-либо вводу с физического устройства.

При вызове. AH = 44H
AL = 02H
BX = дескриптор
CX = число байтов для чтения
DS:DX = сегмент : относительный_адрес буфера

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = число прочитанных байтов, а в буфере находятся управляющие данные из драйвера
Если функция не выполнилась
Флаг переноса = установлен
AX = код ошибки

Замечания. В случае поддержки драйвером эта подфункция может применяться для получения аппаратно-зависимой информации о состоянии, недоступной при обращении к другим функциям MS-DOS.

Драйверам символьно-ориентированных устройств не требуется поддержка IOCTL с подфункцией 02H. Чтобы проверить, поддерживает ли драйвер эту подфункцию, программа может проверить бит 14 информационного слова устройства, который возвращается с помощью IOCTL с подфункцией 00H. Если запрашивается подфункция 02H, а драйвер не способен обработать управляющие данные, то управление возвращается программе с установленным флагом переноса и кодом ошибки 0001H (недопустимая функция) в регистре AX.

Пример. Прочитать из драйвера стандартной печати в буфер buff управляющую строку:

```
stdprn  equ     4               ; Дескриптор стандартной печати
buflen  equ     64              ; Длина буфера

ctllen  dw      ?               ; Длина управляющей строки
buff    db      buflen dup (0)  ; Содержит управляющую строку
        .
        .
        .
        mov     ax,4402h        ; Функция и подфункция
        mov     bx,stdprn       ; Дескриптор стандартной печати
        mov     cx,buflen       ; Длина буфера
        mov     dx,seg buff     ; Адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция чтения не выполнена
        mov     ctllen,ax       ; Сохранить длину управляющей строки
        .
        .
        .
```

Int 21H [2.0]

Функция 44Н(68), подфункция 03Н

IOCTL: Записать управляющие данные в драйвер символьно-ориентированного устройства

Передает управляющие данные из прикладной программы в драйвер символьно-ориентированного устройства. Длина и содержание данных задаются для каждого драйвера устройства без соблюдения какого-либо стандартного формата. Эта функция не обязательно приводит к какому-либо выводу на физическое устройство.

При вызове. AH = 44H
AL = 03H
BX = дескриптор
CX = число байтов для записи
DS:DX = сегмент : относительный_адрес данных

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = число переданных байтов
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. При поддержке драйвером эта подфункция может применяться для запроса аппаратно-зависимых операций (таких, как установка скорости передачи информации в бодах для последовательного порта), что не поддерживается при обращении к другим функциям MS-DOS.

Драйверам символьно-ориентированных устройств не требуется поддерживать IOCTL с подфункцией 03H. Чтобы определить, поддерживает ли драйвер эту подфункцию, программа может проверить бит 14 информационного слова устройства, которое возвращается с помощью IOCTL с подфункцией 00H. Если запрашивается подфункция 03H, а драйвер не способен выполнить управляющие данные, то управление возвращается программе с установленным значением флага переноса и кодом ошибки 0001H (недопустимая функция) в регистре AX.

Пример. Записать из буфера buff в драйвер стандартного устройства печати управляющую строку. Предполагается, что длина строки будет в переменной ctllen:

```
stdprn  equ     4               ; Дескриптор стандартной печати
buflen  equ     64              ; Длина буфера

ctllen  dw      ?               ; Длина управляющих данных
buff    db      buflen dup (?)  ; Содержит управляющие данные
        .
        .
        .
        mov     ax,4403h        ; Функция и подфункция
        mov     bx,stdprn       ; Дескриптор стандартной печати
        mov     dx,seg buff     ; Адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        mov     cx,ctllen       ; Длина управляющих данных
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция вывода записи
                                ; не выполнена
        .
        .
        .
```

Int 21H **[2.0]**

Функция 44H(68), подфункция 04H

IOCTL: Читать управляющие данные из драйвера блочно-ориентированного устройства

Передает управляющие данные из драйвера блочно-ориентированного устройства в буфер прикладной программы. Длина и содержание данных задаются для каждого драйвера устройства без соблюдения какого-то стандартного формата. Эта функция не обязательно приводит к какому-либо вводу из физического устройства.

При вызове. AH = 44H
AL = 04H
BL = код дисковода (0 = по умолчанию, 1 = A, 2 = B и т. д.)
CX = число байтов для чтения
DS:DX = сегмент : относительный_адрес буфера

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = число переданных байтов,
а в буфере находятся управляющие данные из драйвера устройства
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. В случае поддержки драйвером эта подфункция может применяться для получения аппаратно-зависимой информации о состоянии, недоступной при обращении к другим функциям MS-DOS.

Драйверам блочно-ориентированных устройств не требуется поддерживать IOCTL с подфункцией 04H. Если эта подфункция запрашивается, а драйвер не способен выполнить управляющие данные, то управление возвращается программе, при этом установлен флаг переноса и в регистре AX находится код ошибки 0001H (недопустимая функция).

Пример. Прочитать из драйвера блочно-ориентированного устройства, находящегося на дисководе C, в буфер buff управляющую строку:

```
buflen  equ     64              ; Длина буфера

ctllen  dw      ?               ; Длина управляющей строки
buff    db      buflen dup (0)  ; Получает управляющую строку
        .
        .
        .
        mov     ax,4404h        ; Функция и подфункция
        mov     bl,3            ; Дисковод C = 3
        mov     cx,buflen       ; Длина буфера
        mov     dx,seg buff     ; Адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        int     21h             ; Переход в MS-DOS
        jc      error           ; Переход, если операция чтения не выполнена
        mov     ctllen,ax       ; Сохранить длину управляющей строки
        .
        .
        .
```

Int 21H **[2.0]**

Функция 44H(68), подфункция 05H

IOCTL: Записать управляющие данные в драйвер блочно-ориентированного устройства

Передает управляющие данные из прикладной программы непосредственно в драйвер блочно-ориентированного устройства. Длина и содержимое управляющих данных задаются для каждого драйвера устройства без соблюдения какого-то стандартного формата. Эта функция не обязательно приводит к какому-либо выводу на физическое устройство.

При вызове. AH = 44H
AL = 05H
BL = код дисковода (0 = по умолчанию, 1 = A, 2 = B и т. д.)
CX = число байтов для записи
DS:DX = сегмент : относительный_адрес данных

При возврате: Если функция успешно выполнена
Флаг переноса = сброшен
AX = число переданных байтов
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. В случае поддержки драйвером эта подфункция может применяться для запроса аппаратно-зависимых операций (таких, как перемотка магнитной ленты или выбрасывание диска), что не предоставляется с помощью вызова других функций MS-DOS.

Драйверам блочно-ориентированных устройств не требуется поддерживать IOCTL с подфункцией 05H. Если эта подфункция запрашивается, а драйвер не способен обработать управляющие данные, то управление возвращается программе, при этом установлен флаг переноса и в регистре AX находится код 0001H (недопустимая функция).

Пример. Записать из буфера buff в драйвер блочно-ориентированного устройства на дисководе C управляющую строку. Предполагается, что длина строки будет находиться в переменной ctllen:

```
buflen  equ     64              ; Длина буфера

ctllen  dw      ?               ; Длина управляющих данных
buff    db      buflen dup (?)  ; Содержит управляющие данные
        .
        .
        .
        mov     ax,4405h        ; Функция и подфункция
        mov     bl,3            ; Дисковод C = 3
        mov     dx,seg buff     ; Адрес буфера
        mov     ds,dx
        mov     dx,offset buff
        mov     cx,ctllen       ; Длина управляющих данных
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция вывода записи
                                ; не выполнена
```

Int 21H **[2.0]**
Функция 44H(68), подфункция 06H
IOCTL: Проверить состояние ввода

Возвращает код, который указывает, готовы ли к вводу связанные с дескриптором устройство или файл.

При вызове. AH = 44H
AL = 06H
BX = дескриптор

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
и для устройства:
AL = 00H, если устройство не готово
FFH, если устройство готово
или для файла:
AL = 00H, если указатель файла занимает положение на метке EOF
FFH, если указатель файла не занимает положения на метке EOF
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. Эту функцию можно использовать для проверки состояния символьно-ориентированных устройств типа последовательного порта, которые не имеют своей «традиционной» системной функции проверки состояния.

Пример. Проверить, готов ли символ на стандартном дополнительном устройстве (обычно COM1):

```
stdaux  equ     3               ; Дескриптор стандартного дополнительного
                                ; устройства
        .
        .
        .
        mov     ax,4406h        ; Функция и подфункция
        mov     bx,stdaux       ; Дескриптор стандартного дополнительного
                                ; устройства
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        or      al,al           ; Проверить флаг состояния
        jnz     ready           ; Перейти, если символ готов
        .
        .
        .
```

Int 21H **[2.0]**
Функция 44H(68), подфункция 07H
IOCTL: Проверить состояние вывода

Возвращает код, который указывает, готово ли к выводу связанное с дескриптором устройство.

При вызове. AH = 44H
AL = 07H
BX = дескриптор

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
и для устройства:

AL = 00H, если устройство не готово
FFH, если устройство готово
или для файла:
AL = FFH
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. В тех случаях, когда используется дескриптор для файла, эта функция возвращает состояние готовности, даже если диск переполнен или отсутствует в дисководе.

Пример. Проверить, может ли стандартное дополнительное устройство (обычно COM1) принять символ на вывод:

```
stdaux  equ     3               ; Дескриптор стандартного дополнительного
                                ; устройства
        .
        .
        .
        mov     ax,4407h        ; Функция и подфункция
        mov     bx,stdaux       ; Дескриптор стандартного дополнительного
                                ; устройства
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        or      al,al           ; Проверить флаг состояния
        jnz     ready           ; Перейти, если устройство не занято
        .
        .
        .
```

Int 21H **[2.0]**

Функция 44H(68), подфункция 08H

IOCTL: Проверить, является ли блочно-ориентированное устройство сменным

Проверяет, содержит ли заданное блочно-ориентированное устройство сменный магнитный носитель (например, дискету).

При вызове. AH = 44H
AL = 08H
BL = номер дисковода (0 = по умолчанию, 1 = A, 2 = B и т. д.)

При возврате. Если функция успешно завершена
Флаг переноса = сброшен
AL = 00H, если носитель сменный
01H, если носитель не сменный
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Если файл не найден, как ожидалось, на конкретном дисководе, программа может использовать эту подфункцию для определения целесообразности требования к пользователю установки нового диска.

Эта подфункция не может использоваться для сетевого дисковода.

Драйверам блочно-ориентированных устройств не требуется поддерживать подфункцию 08H. Если эта подфункция запрашивается, а блочно-ориентированное устройство не может дать информацию, то управление возвращается программе, при этом

флаг переноса установлен и в регистре AX находится код ошибки 0001H (недопустимая функция).

Пример. Проверить, является ли дисковод C сменным:

```
.
.
.
mov     ax,4408h        ; Функция и подфункция
mov     bl,3            ; Дисковод 3 = C
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если функция не выполнена
and     al,1            ; Проверить тип носителя
jnz     fixed           ; Перейти, если это несъемный диск
.
.
.
```

Int 21H [3.1]

Функция 44H(68), подфункция 09H

IOCTL: Проверить, является ли блочно-ориентированное устройство удаленным

Проверяет, является ли заданное блочно-ориентированное устройство локальным (подключено к компьютеру, выполняющему программу) или удаленным (перенаправлено на сервер сети).

При вызове. AH = 44H
AL = 09H
BL = номер дисковода (0 = по умолчанию, 1 = A, 2 = B и т. д.)

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
DX = слово атрибута устройства
бит 12 = 0, если дисковод локальный
1, если дисковод удаленный
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. Следует избегать применения этой подфункции. Прикладные программы не должны различать файлы на локальных и удаленных устройствах.

Пример. Проверить, смонтирован ли дисковод D на машине, выполняющей программу, или это сетевой дисковод:

```
.
.
.
mov     ax,4409h        ; Функция и подфункция
mov     bl,4            ; Дисковод 4 = D
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если функция не выполнена
and     dx,1000h        ; Проверить бит локальный/удаленный
jnz     remote          ; Перейти, если это сетевой дисковод
.
.
.
```

Int 21H [3.1]
Функция 44H(68), подфункция 0AH(10)
IOCTL: Проверить, является ли дескриптор удаленным

Проверяет, отсылает ли заданный дескриптор к локальному файлу или устройству (расположенному на компьютере, который выполняет программу) или удаленному (находящемуся на сервере сети).

При вызове. AH = 44H
AL = 0AH
BX = дескриптор

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
DX = слово атрибута файла или устройства
бит 15 = 0, если локальный
1, если удаленный
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Обычно прикладные программы не должны пытаться различать файлы на локальных и удаленных устройствах.

Если сеть не начала работать, то управление возвращается в вызывающую программу, при этом флаг переноса установлен и в регистре AX находится код ошибки 0001H (недопустимая функция).

Пример. Проверить, связан ли дескриптор, сохраненный в переменной fhandle, с файлом или устройством, расположенным на машине, выполняющей программу, или на сервере сети:

```
fhandle dw      ?               ; Дескриптор устройства
        .
        .
        .
        mov     ax,440ah        ; Функция и подфункция
        mov     bx,fhandle      ; Дескриптор файла или устройства
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        and     dx,800h         ; Проверить бит локальный/удаленный
        jnz     remote          ; Перейти, если сетевой дескриптор
        .
        .
        .
```

Int 21H [3.1]
Функция 44H(68), подфункция 0BH(11)
IOCTL: Изменить счетчик повторных попыток разделения

Устанавливает, сколько раз система MS-DOS повторяет дисковую операцию, после того как она не выполнилась вследствие нарушения условий разделения файла, прежде чем вернуть код ошибки в процесс, активизировавший этот запрос. Эта подфункция недоступна, пока не загружен модуль разделения файла (SHARE.EXE).

При вызове. AH = 44H
AL = 0BH
CX = задержка каждой повторной попытки (по умолчанию = 1)
DX = число повторных попыток (по умолчанию = 3)

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Задержка – величина, зависящая от компьютера, и определяется типом центрального процессора, а также тактовой частотой таймера. Каждая задержка определяется следующей последовательностью команд:

```
xor     cx,cx
loop    $
```

которая выполняется 65536 раз, прежде чем покинуть цикл.

Счетчик повторных попыток разделения влияет на поведение всей системы в целом и не является локальным параметром процесса. Если программа изменяет содержимое счетчика повторных попыток разделения, то до завершения своей работы она должна восстановить исходное содержимое счетчика, записав в него значение, действующее по умолчанию.

Пример. Изменить на 5 число автоматических повторных попыток в случае нарушения условий разделения файла:

```
.
.
mov     ax,440bh        ; Функция и подфункция
mov     cx,1            ; Задержка после каждого повтора
mov     dx,5            ; Число повторов
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если функция не выполнена
.
.
.
```

Int 21H [3.2]
Функция 44H(68), подфункция 0CH(12)
IOCTL: Родовое управление вводом-выводом символьно-ориентированных устройств

Предоставляет универсальный механизм взаимодействия между прикладными программами и драйверами символьно-ориентированных устройств.

При вызове. AH = 44H
AL = 0CH
BX = дескриптор
CH = старший код (код категории)
00H = неизвестное устройство
01H = COM1, COM2, COM3 или COM4 (3.3)
03H = CON (клавиатура и дисплей) (3.3)
05H = LPT1, LPT2 или LPT3 (3.2)
CL = младший код (код функции)
45H = установить счетчик числа итераций (3.2)
4AH = выбрать кодовую страницу (3.3)
4CH = начать подготовку кодовой страницы (3.3)
4DH = закончить подготовку кодовой страницы (3.3)
5FH = установить информацию о дисплее (4.0)
65H = прочитать счетчик числа итераций (3.2)

6AH = запросить выбранную кодовую страницу (3.3)
6BH = запросить подготовленную печать (3.3)
7FH = получить информацию о дисплее (4.0)
DS:DX = сегмент: относительный_адрес блока параметров

При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
и, если при вызове CL = 65H, 6AH, 6BH или 7FH
DS:DX = сегмент : относительный_адрес блока параметров
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. В тех случаях, когда младший код равен 45H (Установить счетчик числа итераций) или 65H (Прочитать счетчик числа итераций), блок параметров представляет собой просто 2-байт буфер, содержащий или получающий число итераций для принтера. Такое обращение допустимо только для устройств печати, поддерживающих вывод до занятости, и определяет, сколько тактов драйвер устройства будет ожидать от устройства сигнала готовности до возврата из вызова на вывод.

Блок параметров для младшего кода 4DH (закончить составление кодовой страницы) имеет следующий формат:

```
dw      2               ; Длина последующих данных
dw      0               ; ( зарезервировано )
```

В среде MS-DOS версии 3.3 в случае равенства младшего кода 4AH (Выбрать кодовую страницы) или 6AH (Запросить выбранную кодовую страницу) блок параметров имеет следующий формат:

```
dw      2               ; Длина последующих данных
dw      ?               ; Идентификатор кодовой страницы
```

В среде MS-DOS версии 4.0 младший код, равный 4AH или 6AH, также устанавливает или получает двухбайтовый набор символов (double-byte character set, DBCS) таблицы лидирующих байтов и, кроме того, используется следующий формат:

```
dw      (n+2)*2+1       ; Длина последующих данных
dw      ?               ; Идентификатор кодовой страницы
db      start,end       ; Лидирующие байты области 1 DBCS
.
.
.
db      start,end       ; Лидирующие байты области n DBCS
db      0,0
```

Блок параметров для младшего кода 4CH (Начать подготовку кодовой страницы) имеет следующий формат:

```
dw      0               ; Тип шрифта
                        ; Бит 0 = 0 загружен
                        ;       = 1 кассета
                        ; Биты 1 - 15 = зарезервированы (0)
dw      (n+1)*2         ; Длина остатка
                        ; блока параметров
```

```
dw      n               ; Число кодовых страниц
                        ; последующей таблицы
dw      ?               ; Кодовая страница 1
dw      ?               ; Кодовая страница 2
.
.
.
dw      ?               ; Кодовая страница n
```

Блок параметров для младшего кода 6BH (Запросить подготовленную печать) имеет следующий формат, предполагая n аппаратных кодовых страниц и m подготовленных кодовых страниц ($n \leqslant 12$, $m \leqslant 12$):

```
dw      (n+m+2)*2       ; Длина последующих данных
dw      n               ; Число аппаратных кодовых страниц
dw      ?               ; Кодовая аппаратная страница 1
dw      ?               ; Кодовая аппаратная страница 2
.
.
.
dw      ?               ; Кодовая аппаратная страница n
dw      m               ; Число подготовленных кодовых страниц
dw      ?               ; Подготовленная кодовая страница 1
dw      ?               ; Подготовленная кодовая страница 2
.
.
.
dw      ?               ; Подготовленная кодовая страница n
```

После вызова младшего кода 4CH (Начать подготовку кодовой страницы) данные, определяющие шрифт кодовой страницы, записываются в драйвер с помощью одного или более вызовов функции «Управление вводом-выводом» с подфункцией «Записать управляющие данные» (прерывание Int 21H с функцией 44H, подфункцией 03H). Формат данных зависит от драйвера и устройства. После того как данные шрифта записаны в драйвер, следует задать обращение к младшему коду 4DH (Закончить подготовку кодовой страницы). Если между вызовами младшего кода 4CH и 4DH в драйвер не записаны какие-нибудь данные, то драйвер интерпретирует вновь подготовленные кодовые страницы как аппаратные.

Специальный вариант младшего кода 4CH (Начать подготовку кодовой страницы), называемый «регенерация», требуется для того, чтобы фактически загрузить в периферийное устройство подготовленные кодовые страницы. Операция регенерации выполняется по запросу с младшим кодом 4CH при наличии — 1 во всех позициях кодовой страницы в блоке параметров с последующим вызовом с младшим кодом 4DH (Закончить подготовку кодовой страницы).

[4.0+] Для младших кодов 5FH (Установить информацию о дисплее) и 7FH (Получить информацию о дисплее) блок параметров форматируется следующим образом:

```
db      0               ; Уровень (0 в системе MS-DOS 4.0)
db      0               ; Зарезервировано ( должен быть 0 )
dw      14              ; Длина последующих данных
dw      ?               ; Управляющие флаги
                        ; Бит 0       = 0 яркость
                        ;             = 1 мерцание
                        ; Биты 1 - 15 = зарезервированы (0)
```

```
        db      ?               ; Тип режима (1 = текст, 2 = APA )
        db      0               ; Зарезервировано (должен быть 0 )
        dw      ?               ; Цвета
                                ; 0 = монохром
                                ; 1 = 2 цвета
                                ; 2 = 4 цвета
                                ; 4 = 16 цветов
                                ; 8 = 256 цветов
        dw      ?               ; Столбцы пиксела
        dw      ?               ; Строки пиксела
        dw      ?               ; Столбцы символа
        dw      ?               ; Строки символа
```

Пример. Получить для устройства стандартной печати текущую кодовую страницу:

```
stdprn  equ     4               ; Дескриптор стандартной печати

pars    dw      2               ; Длина данных
        dw      ?               ; Получает кодовую страницу
        .
        .
        .
        mov     ax,440ch        ; Функция и подфункция
        mov     bx,stdprn       ; Дескриптор стандартной печати
        mov     ch,5            ; Категория LPTx
        mov     cl,6ah          ; Запросить кодовую страницу
        mov     dx,seg pars     ; Адрес блока параметра
        mov     ds,dx
        mov     dx,offset pars
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        .
        .
        .
```

Int 21H **[3.2]**

Функция 44H, подфункция 0DH(13)

IOCTL: Родовое управление вводом-выводом блочно-ориентированных устройств

Предоставляет универсальный механизм взаимодействия между прикладными программами и драйверами блочно-ориентированных устройств. Дает возможность программе проверять или изменять параметры устройства для логического диска, а также читать, записывать, форматировать и проверять дорожки диска аппаратно-независимым образом.

При вызове. AH = 44H
AL = 0DH
BL = код дисковода (0 = по умолчанию, 1 = A, 2 = B и т. д.)
CH = (старший) код категории
08H = дисковод
CL = (младший) код функции
40H = установить параметры внешнего устройства
41H = записать дорожку
42H = форматировать и проверить дорожку
47H = установить флаг доступа (4.0)
60H = получить параметры внешнего устройства
61H = прочитать дорожку

62H = проверить дорожку
67H = получить флаг доступа (4.0)
DS:DX = сегмент : относительный_адрес блока параметров

При возврате: Если функция успешно выполнена
Флаг переноса = сброшен
и, если при вызове CL = 60H или 61H
DS:DX = сегмент : относительный_адрес блока параметров
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Перед попыткой выполнить запись, чтение, форматирование или проверку дорожки логического диска следует испюльзовать функцию младшего кода 40H (Установить параметры внешнего устройства). Как правило, при реализации любой из этих операций используется следующий порядок действий:

- получить текущие параметры (младший код 60H), проверить и сохранить их;
- установить новые параметры (младший код 40H);
- выполнить поставленную задачу;
- извлечь исходные параметры и восстановить их с помощью младшего кода 40H.

Для значений младшего кода 40H (Установить параметры внешнего устройства) и 60H (Получить параметры внешнего устройства) блок параметра форматируется следующим образом:

Поле специальных функций : относительный адрес 00H, длина = 1 байт

Бит (ы)	Значение	Описание
0	0	поле BPB устройства содержит новый BPB, используемый по умолчанию
	1	использовать текущий BPB
1	0	использовать все поля в блоке параметров
	1	использовать только поле формата дорожки
2	0	секторы на дорожке могут быть разных размеров (этого всегда следует избегать)
	1	секторы на дорожке все одного размера; диапазон изменения номера сектора от 1 до фактического числа секторов на дорожке (всегда должно быть так)
3–7	0	зарезервированы

Поле типа устройства : относительный адрес 01H, длина 1 байт

Значение	Описание
0	320/360 Кбайт, диск 5,25 дюйма
1	1,2 Мбайт, диск 5,25 дюйма
2	720 Кбайт, диск 3,5 дюйма
3	одинарная плотность, диск 8 дюймов
4	двойная плотность, диск 8 дюймов
5	жесткий диск
6	накопитель на магнитной ленте
7	другой тип блочно-ориентированного устройства

Поле атрибута устройства : относительный адрес 02H, длина = 1 слово

Бит (ы)	Значение	Описание
0	0	сменный магнитный накопитель
	1	постоянный магнитный накопитель
1	0	запирание дверцы не поддерживается
	1	запирание дверцы поддерживается
2–15	0	зарезервированы

Поле числа цилиндров: относительный адрес 04H, длина 1 слово
Максимальное число цилиндров, поддерживаемое на блочно-ориентированном устройстве.
Поле типа носителя: относительный адрес 06H, длина 1 байт

Значение	Описание
0	1,2 Мбайт, диск 5,25 дюйма
1	320/360 Кбайт, диск 5,25 дюйма

Поле BPB устройства: относительный адрес, длина = 31 байт
О формате BPB-устройства см. ниже отдельное замечание.
Если в поле специальных функций бит 0 = 0, то это поле содержит новый BPB устройства, используемый по умолчанию.
Если в поле специальных функций бит 0 = 1, то BPB возвращается в это поле с помощью драйвера устройства в ответ на последующие запросы «Построить BPB».
Поле формата дорожки: относительный адрес 26H, таблица переменной длины

Длина	Описание
Слово	число секторов на дорожке
Слово	номер первого сектора на дорожке
Слово	размер первого сектора на дорожке
.	
.	
.	
Слово	номер последнего сектора на дорожке
Слово	размер последнего сектора на дорожке.

Поле BPB-устройства – это структура данных длиной в 31 байт, которая описывает текущий диск и его области управления. Поле форматируется следующим образом:

Байт (ы)	Описание
00H–01H	число байтов на секторе
02H	число секторов на кластер (единица размещения)
03H–04H	зарезервированные секторы, начиная с сектора 0
05H	число таблиц размещения файлов (FAT)
06H–07H	максимальное число записей в корневом каталоге
08H–09H	число секторов
0AH	идентификатор магнитного носителя
0BH–0CH	число секторов на FAT
0DH–0EH	число секторов на дорожку
0FH–10H	число головок
11H–14H	число скрытых секторов
15H–18H	максимальное число секторов (если байты 08H–09H = 0)
19H–1EH	зарезервированы

При использовании младшего кода 40H (Установить параметры внешнего устройства) число цилиндров не должно изменяться, в противном случае или некоторые, или все тома могут стать недоступными.

Для младших кодов 41H (Записать на трек) и 61H (Прочитать трек) блок параметров форматируется следующим образом:

Байт (ы)	Описание
00H	поле специальных функций (должен быть 0)
01H–02H	головка
03H–04H	цилиндр
05H–06H	начальный сектор
07H–08H	секторы для передачи
09H–0CH	адрес буфера передачи

Для значений младшего кода 42H (Форматировать и проверить дорожку) и 62H (Проверить дорожку) блок параметров форматируется следующим образом:

Байт (ы)	Описание
00H	поле специальных функций
	Бит (ы) — *Описание*
	0 — 0 = форматировать/проверить дорожку 1 = вызвать состояние формата (только для MS-DOS 4.0)
	1–7 — зарезервированы (0)
01H–02H	головка
03H–04H	цилиндр

Для определения того, поддерживает ли драйвер заданное число дорожек и секторов на дорожку, в среде MS-DOS 4.0 можно вызвать данную функцию с установленным битом 0 поля специальных функций после вызова младшего кода 40H (Установить параметры внешнего устройства). Состояние возвращается в поле специальных функций, которое интерпретируется следующим образом:

Значение	Описание
0	заданное число дорожек и секторов на дорожку поддерживается
1	эта функция не поддерживается ПЗУ BIOS
2	заданное число дорожек и секторов на дорожку не поддерживается
3	на дисководе нет диска

Для младших кодов 47H (Установить флаг доступа) и 67H (Получить флаг доступа) блок параметров форматируется следующим образом:

Байт	Описание
00H	поле специальных функций (должен быть 0)
01H	флаг доступа диска

В тех случаях, когда флаг доступа диска = 0, доступ к магнитному носителю блокируется драйвером. Флаг доступа диска = 0 в тех случаях, когда драйвер обнаруживает неформатированный магнитный носитель или носитель с недопустимой загрузочной записью. В тех случаях, когда флаг доступа отличен от 0, операции чтения-записи с магнитным носителем допускаются драйвером. Программа форматирования

должна сбрасывать флаг доступа диска с помощью младшего кода 47H до того, как она сделает запрос к младшему коду 42H (Форматировать и проверить трек).
Пример. Получить блок параметров внешнего устройства на дисководе C:

```
dbpb    db      128 dup (0)     ; Блок параметров устройства
        .
        .
        .
        mov     ax,440dh        ; Функция и подфункция
        mov     bl,3            ; Дисковод C = 3
        mov     ch,8            ; Тип диска
        mov     cl,60h          ; Получить параметры устройства
        mov     dx,seg dbpb     ; Адрес буфера
        mov     ds,dx
        mov     dx,offset dbpb
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        .
        .
        .
```

Int 21H [3.2]
Функция 44H(68), подфункция 0EH(14)
IOCTL: Получить карту логического дисковода

Возвращает код логического дисковода, который последний раз использовался для доступа к заданному блочно-ориентированному устройству.
При вызове. AH = 44H
AL = 0EH
BL = код дисковода (0 = по умолчанию, 1 = A, 2 = B и т. д.)
При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AL = код отображения
00H, если блочно-ориентированному устройству назначен только один код логического дисковода
01H–1AH код логического дисковода (1 = A, 2 = B и т. д.), отображенный на блочно-ориентированное устройство
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки
Замечание. Если дисководу не было назначено логическое отображение с помощью функции 44H с подфункцией 0FH, то коды логического и физического дисковода совпадают.
Пример. Проверить, имеет ли дисковод A более одного кода логического дисковода.

```
        .
        .
        .
        mov     ax,440eh        ; Функция и подфункция
        mov     bl,1            ; Дисковод 1 = A
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        or      al,al           ; Проверить код дисковода
        jz      label1          ; Перейти, если нет другого имени дисковода
        .
        .
        .
```

Int 21H [3.2]
Функция 44H(68), подфункция 0FH(15)
IOCTL: Установить карту логического дисковода

Устанавливает следующий код логического дисковода, который будет использоваться для ссылки к блочно-ориентированному устройству.
При вызове. AH = 44H
AL = 0FH
BL = код дисковода (0 = по умолчанию, 1 = A, 2 = B и т. д.)
При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AL = код отображения
00H, если только один код логического дисковода назначен блочно-ориентированному устройству
01H–1AH код логического дисковода (1 = A, 2 = B и т. д.), отображенный на блочно-ориентированное устройство
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки
Замечание. В тех случаях, когда физическое блочно-ориентированное устройство описывается более чем одним кодом логического дисковода, данную функцию можно использовать для передачи драйверу информации о том, какой следующий код дисковода будет использоваться в процессе доступа к устройству.
Пример. Сообщить драйверу дискеты, что следующее обращение будет к логическому дисководу B:

```
.
.
.
mov     ax,440fh        ; Функция и подфункция
mov     bl,2            ; Дисковод 2 = B
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если функция не выполнена
.
.
.
```

Int 21H [2.0]
Функция 45H(69)
Дублировать дескриптор

Получив дескриптор открытого в настоящий момент устройства или файла, возвращает новый дескриптор, который указывает то же устройство или ту же позицию в файле.
При вызове. AH = 45H
BX = дублируемый дескриптор
При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = новый дескриптор
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки
Замечания. Операции поиска, чтения и записи, изменяющие положение указателя

файла для одного из двух дескрипторов, одновременно изменяют положение указателя файла, связанного с другим дескриптором.

Эту функцию можно эффективно использовать в целях обновления в каталоге записи для файла, который изменил длину, без издержек на закрытие и последующее повторное открытие файла. С помощью данной функции попросту дублируется дескриптор файла, и дубликат закрывается, причем исходный дескриптор остается открытым для последующих операций чтения-записи.

[3.3] См. также прерывание Int 21H с функцией 68H (Дополнить файл).

Пример. Дублировать хранящийся в переменной fhandle дескриптор, после чего закрыть дубликат. Это гарантирует, что все буферизованные данные физически записаны на диск, запись каталога для соответствующего файла обновлена и оставляет исходный дескриптор открытым для последующих файловых операций:

```
fhandle dw      0               ; Дескриптор файла
        .
        .
        .
                                ; Получить дубликат дескриптора
        mov     ah,45h          ; Номер функции
        mov     bx,fhandle      ; Дескриптор исходного файла
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если дублирование
                                ; не выполнено
        mov     bx,ax           ; Поместить дескриптор в BX
        mov     ah,3eh          ; Номер функции
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция закрытия
                                ; не выполнена
```

Int 21H **[2.0]**
Функция 46H(70)
Перенаправить дескриптор

Получив два дескриптора, заставляет второй дескриптор указывать на то же устройство или ту же позицию в файле, что и первый дескриптор. В этом случае говорят, что второй дескриптор перенаправлен.

При вызове. AH = 46H
BX = дескриптор файла или устройства
CX = перенаправляемый дескриптор

При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Если переданный в CX дескриптор уже указывает на открытый файл, то этот файл сначала закрывается. Операции поиска, чтения и записи, изменяющие положение указателя файла для одного из двух дескрипторов, одновременно изменяют положение указателя файла, связанного с другим дескриптором.

Обычно данная функция применяется с целью перенаправить дескрипторы стандартного ввода или вывода на другой файл или другое устройство, прежде чем дочерний процесс запустится с помощью прерывания Int 21H с функцией 4BH.

Пример. Перенаправить стандартный вывод на устройство печати, так что весь вывод, направляемый на консоль, вместо этого будет появляться на принтере. Позднее восстановить исходное значение дескриптора стандартного вывода:

```
stdin   equ     0
stdout  equ     1
stderr  equ     2
stdaux  equ     3
stdprn  equ     4

dhandle dw      0               ; Дубликат дескриптора
        .
        .
        .
                                ; Получение дубликата стандартного
                                ; вывода
        mov     ah,45h          ; Номер функции
        mov     bx,stdout       ; Дескриптор стандартного вывода
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если дублирование не выполнено
        mov     dhandle,ax      ; Сохранить дубликат дескриптора
                                ;
                                ; Перенаправление стандартного вывода на
                                ; устройство стандартной печати
        mov     ah,46h          ; Номер функции
        mov     bx,stdprn       ; Дескриптор стандартной печати
        mov     cx,stdout       ; Дескриптор стандартного вывода
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция перенаправления
                                ; не выполнена



                                ; Восстановление исходного значения
                                ; стандартного вывода
        mov     ah,dhandle      ; Номер функции
        mov     cx,stdout       ; Сохраненный дубликат дескриптора
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция перенаправления
                                ; не выполнена
                                ; Закрытие дубликата дескриптора,
                                ; т.к. необходимость в нем отпала
        mov     ah,3eh          ; Номер функции
        mov     bx,dhandle      ; Сохраненный дубликат дескриптора
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция закрытия
                                ; не выполнена
```

Int 21H **[2.0]**
Функция 47H(71)
Получить текущий каталог

Получает строку символов в коде ASCIIZ, которая описывает путь от корневого каталога к текущему, а также имя текущего каталога.
При вызове. AH = 47H

DL = код дисковода (0 = по умолчанию, 1 = A и т. д.)
DS:SI = сегмент : относительный_адрес буфера в 64 байт

При возврате. Если функция выполнена успешно
Флаг переноса = сброшен,
а буфер заполнен описанием полного пути от корневого каталога до текущего
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Возвращаемое описание пути не содержит идентификатора дисковода и начального символа «\». Описание пути заканчивается нулевым (00H) байтом. Следовательно, если текущий каталог является корневым, то первый байт в буфере будет содержать 00H.

Функция не выполняется в случае недопустимого кода дисковода.

Установить текущий каталог можно с помощью прерывания Int 21H с функцией 3BH.

Пример. Прочитать в буфер dbuff имя текущего каталога для дисковода C:

```
dbuff   db      64 dup (0)      ; Получает строку с указанием пути
        .
        .
        .
        mov     ah,47h          ; Номер функции
        mov     dl,03           ; Дисковод C = 3
        mov     si,seg dbuff    ; Адрес буфера
        mov     ds,si
        mov     si,offset dbuff
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти в случае ошибки
        .
        .
        .
```

Int 21H **[2.0]**
Функция 48H(72)
Выделить блок памяти

Выделяет блок памяти и возвращает указатель начала выделенной области.

При вызове. AH = 48H
BX = число параграфов необходимой памяти

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = адрес базового сегмента выделенной области
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки
BX = размер наибольшего доступного блока (в параграфах)

Замечания. Если функция выполнена успешно, то адрес базы вновь выделенного блока – AX : 0000.

Применяемая системой MS-DOS по умолчанию стратегия выделения – «поиск от начала», т. е. выделяется блок памяти с наименьшим адресом, размер которого достаточно большой, чтобы удовлетворить запрос. С помощью прерывания Int 21H с функцией 58H можно изменить стратегию выделения.

В тех случаях, когда загружается программа типа .COM, то обычно она уже владеет всей памятью в области транзитных программ, ничего не оставляя для динамического выделения. Размер памяти, первоначально выделяемой в момент загрузки программ типа .EXE, зависит от полей MINALLOC и MAXALLOC в заголовке файла типа .EXE. (См. прерывание Int 21H с функцией 4AH.)

Пример. Запросить память размером 64 Кбайт для использования ее в качестве буфера:

```
bufseg  dw      ?               ; База сегмента нового блока
        .
        .
        .
        mov     ah,48h          ; Номер функции
        mov     bx,1000h        ; Размер блока (в параграфах)
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти,если операция выделения
                                ; не выполнена
        mov     bufseg,ax       ; Сохранить сегмент нового блока
        .
        .
        .
```

Int 21H **[2.0]**
Функция 49H(73)
Освободить блок памяти

Освобождает блок памяти и делает его доступным для использования другими программами.

При вызове. AH = 49H
ES = сегмент освобождаемого блока

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Эта функция предполагает, что блок освобождаемой памяти ранее был получен с помощью вызова прерывания Int 21H с функцией 48H.

Данная функция не будет выполняться или может вызвать непредсказуемые системные ошибки, если:

– программа освобождает блок памяти, который ей не принадлежит;

– переданный в регистр ES адрес сегмента – это недопустимый адрес базы для существующего блока памяти.

Пример. Освободить блок памяти, ранее выделенный в примере для прерывания Int 21H с функцией 48H:

```
bufseg  dw      ?               ; База сегмента блока
        .
        .
        .
        mov     ah,49h          ; Номер функции
        mov     es,bufseg       ; База сегмента блока
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция освобождения блока
                                ; памяти не выполнена
```

Int 21H **[2.0]**
Функция 4AH(74)
Изменить размер блока памяти

Динамически сокращает или расширяет блок памяти в соответствии с потребностями прикладной программы.

При вызове. AH = 4AH
BX = желательный размер нового блока в параграфах
ES = сегмент модифицируемого блока

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки
BX = максимальный доступный размер блока (в параграфах)

Замечания. Данная функция модифицирует размер блока памяти, ранее выделенного через вызов прерывания Int 21H с функцией 48H.

Если в программе запрашивается увеличение размера выделенного блока, а эта функция не выполняется, то в регистр BX возвращается максимально возможный размер заданного блока. Программа может использовать это значение для того, чтобы определить, следует ли ей завершиться или можно продолжить выполнение в условиях нехватки памяти.

Программа, которая использует функцию EXEC (прерывание Int 21H с функцией 4BH) для загрузки и выполнения дочерней программы, сначала должна обратиться к данной функции с тем, чтобы сделать память доступной для дочернего процесса путем передачи адреса своего PSP в регистр ES и размера необходимой для своего выполнения памяти (включая саму программу, данные и стек) в регистр BX.

Пример. Изменить размер блока памяти, который был выделен в примере для прерывания Int 21H с функцией 48H, уменьшив его до 32 Кбайт:

```
bufseg  dw      ?               ; База сегмента блока
        .
        .
        .
        mov     ah,4ah          ; Номер функции
        mov     bx,0800h        ; Новый размер блока (в параграфах)
        mov     es,bufseg       ; База сегмента блока
        int     21h             ; Переход в MS-DOS
        jc error                ; Перейти,если операция изменения
                                ; размера блока памяти не выполнена
```

Int 21H **[2.0]**
Функция 4BH(75)
Выполнить программу (EXEC)

Позволяет прикладной программе выполнить другую программу, после завершения которой вновь получить управление. Кроме того, может использоваться для загрузки оверлейных программ, хотя это используется редко.

При вызове. AH = 4BH
AL = подфункция

00H = загрузить и выполнить программу
03H = загрузить оверлей
ES:BX = сегмент : относительный_адрес блока параметров
DS:DX = сегмент : относительный_адрес описания пути к программе, заданного в коде ASCIIZ

При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
[2] Все регистры, за исключением CS:IP, могут быть разрушены
[3.0+] Регистры сохраняются обычным образом
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Формат блока параметров для подфункции 00H (Загрузить и выполнить программу) имеет вид:

Байты	Описание
00H–01H	сегментный указатель окружения блока
02H–03H	относительный адрес хвоста командной строки
04H–05H	сегмент хвоста командной строки
06H–07H	относительный адрес первого FCB, копируемого в новый префикс по адресу PSP + 5CH
08H–09H	сегмент первого FCB
0AH–0BH	относительный адрес второго FCB, копируемого в новый префикс по адресу PSP + 6CH
0CH–0DH	сегмент второго FCB

Формат блока параметров для подфункции 03H (Загрузить оверлей) имеет следующий вид:

Байты	Описание
00H–01H	адрес сегмента, куда должен быть загружен оверлей
02H–03H	константа настройки для загрузочного модуля

Блок окружения должен выравниваться на границу параграфа. Он содержит строки ASCIIZ-кодов в виде

```
db      'COMSPEC = A:\COMMAND.COM',0
```

Вводимый набор строк заканчивается дополнительным нулевым (00H) байтом

Формат хвоста командной строки предусматривает счетчик числа байтов и следующую за ним строку ASCII-кодов, заканчивающуюся кодом возврата каретки (который не учитывается в значении счетчика). Первым символом строки должен быть ASCII-код пробела (20H) для совместимости с хвостом командной строки, пересылаемым в программы программой COMMAND.COM, например:

```
db      6,' *.DAT',0dh
```

Перед вызовом прерывания Int 21H с функцией 4B с целью активизации другой программы текущая программа должна освободить всю фактически не используемую память с помощью прерывания Int 21H с функцией 4AH, передавая при этом адрес сегмента ее собственного PSP и число используемых параграфов.

Обычно все активные дескрипторы родительской программы наследуются дочерней программой, хотя в среде MS-DOS версий 3.0 и старше родительская программа может препятствовать этому, устанавливая бит наследования в тех случаях, когда файл или устройство открыты. Кроме того, всякое перенаправление в родительском процессе стандартного ввода и/или вывода влияет на дочерний процесс.

Блок окружения может использоваться для передачи информации в дочерний процесс. Если указатель блока окружения в блоке параметров равен нулю, то дочерняя программа наследует точную копию родительского окружения. Во всяком случае, адрес сегмента дочернего окружения находится по относительному адресу 002CH PSP дочерней программы.

После возврата из обращения к функции EXEC с помощью прерывания Int 21H с функцией 4DH можно получить тип завершения и код возврата дочерней программы.

Пример. См. гл. 12.

Int 21H **[2.0]**
Функция 4CH(76)
Завершить процесс с кодом возврата

Завершает текущий процесс с передачей кода возврата в родительский процесс. Это один из нескольких методов, которые могут использоваться в программе для осуществления окончательного выхода. В этих случаях системой MS-DOS производятся следующие действия:

- Освобождается вся память, принадлежащая процессу.
- Сбрасываются файловые буферы, а открытые самим процессом дескрипторы файлов или устройств закрываются.
- Восстанавливается по содержимому PSP:000AH вектор обработчика завершения (прерывания Int 22H).
- Восстанавливается по содержимому PSP:000EH вектор обработчика Ctrl-C (прерывание Int 23H).
- [2.0+] Восстанавливается по содержимому PSP:0012H вектор обработчика критической ошибки (прерывание Int 24H).
- Управление передается обработчику завершения.

Если программа возвращается в программу COMMAND.COM, то управление передается резидентной части последней, а ее транзитная часть при необходимости перезагружается. В случае выполнения командного файла выбирается и интерпретируется следующая строка файла; в противном случае выдается запрос на ввод следующей команды пользователя.

При вызове. AH = 4CH
AL = код возврата

При возврате. Ничего не возвращается.

Замечания. [2.0+] Это наиболее предпочтительный метод завершения прикладных программ, так как он позволяет передавать в родительскую программу код завершения и не полагается при этом на содержимое сегментных регистров. Другие способы выполнения окончательного выхода:

- с помощью прерывания Int 20H;
- с помощью прерывания Int 21H с функцией 00H;
- с помощью прерывания Int 21H с функцией 31H;
- с помощью прерывания Int 27H.

Все файлы, открытые с помощью функций FCB-типа и модифицированные программой, необходимо закрыть до завершения работы программы; в противном случае данные могут быть потеряны.

Код возврата может быть извлечен родительским процессом с помощью прерывания Int 21H с функцией 4DH (Получить код возврата). Кроме того, код можно проверить в командном файле с помощью оператора IF ERRORLEVEL. По соглашению значение кода возврата нуль указывает на успешное выполнение функции, а ненулевое значение – на ошибку.

[3.0+] Если программа выполняется в сети, то до завершения программы необходимо освободить все запертые файловые районы.

Пример. Завершить текущий процесс, при этом передать в родительский процесс код возврата, равный 1:

```
        .
        .
        .
        mov     ah,4ch          ; Номер функции
        mov     al,01h          ; Код завершения
        int     21h             ; Переход в MS-DOS
```

Int 21H **[2.0]**
Функция 4DH(77)
Получить код возврата

Применяется родительским процессом после успешного выполнения вызова EXEC (с помощью прерывания Int 21H с функцией 4BH) в целях получения кода возврата и типа завершения дочернего процесса.

При вызове. AH = 4DH

При возврате. AH = тип завершения
- 00H в случае нормального завершения с помощью прерываний Int 20H, Int 21H с функцией 00H или Int 21H с функцией 4CH
- 01H в случае завершения по вводу пользователем Ctrl-C
- 02H в случае завершения с помощью обработчика критической ошибки
- 03H в случае завершения с помощью прерываний Int 21H с функцией 31H и Int 27H

AL = код возврата, переданный дочерним процессом (0, если этот процесс завершен с помощью прерываний Int 20H, Int 21H с функцией 00H или Int 27H)

Замечания. Данная функция будет выдавать код возврата дочернего процесса только один раз. Последующий вызов без промежуточного вызова EXEC (с помощью прерывания Int 21H с функцией 4BH) не обязательно будет возвращать имеющую смысл информацию.

Данная функция не устанавливает флаг переноса для указания ошибки. Если перед этим дочерних процессов не выполнялось, то возвращаемые в регистрах AL и AH значения будут не определены.

Пример. Получить код возврата и тип завершения дочернего процесса, ранее активизированного с помощью прерывания Int 21H с функцией 4BH (EXEC):

```
retcode dw      ?               ; Код возврата, тип завершения
        .
        .
        .
        mov     ah,4dh          ; Номер функции
        int     21h             ; Переход в MS-DOS
```

```
mov     retcode,ax      ; Сохранить информацию о дочернем процессе
.
.
.
```

Int 21H **[2.0]**
Функция 4EH(78)
Найти первый файл

Для данной в виде ASCIIZ-строки спецификации файла ищет в заданном или действующем по умолчанию каталоге на заданном или действующем по умолчанию дисководе первый соответствующий файл.

При вызове. AH = 4EH
CX = атрибут поиска (биты могут комбинироваться)

Бит (ы)	**Описание** (если бит установлен)
0	только для чтения
1	скрытый
2	системный
3	метка тома
4	каталог
5	атрибут архива
6–15	зарезервированы (0)

DS:DX = сегмент : относительный адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена (найден соответствующий файл)
Флаг переноса = сброшен
и результаты поиска возвращены в область передачи текущего диска следующим образом:

Байт (ы)	**Описание**
00H–14H	зарезервированы
15H	атрибут соответствующего файла или каталога
16H–17H	время создания файла биты 00H–04H = 2-секундные приращения (0–29) биты 05H–0AH = минуты (0–59) биты 0BH–0FH = год (относительно 1980 г.)
1AH–1DH	размер файла
1EH–2AH	имя файла и расширение в коде ASCIIZ

Если функция не выполнена (нет соответствующих файлов)
Флаг переноса = установлен
AX = код ошибки

Замечания. Данная функция предполагает, что в программе с помощью прерывания Int 21H с функцией 1AH ранее установлена область передачи диска с буфером соответствующего размера.

В имени файла допускается использование шаблонов групповых операций «×» и «?». Если шаблоны групповых операций присутствуют, то данная функция возвращает только первый файл с подходящим именем. Если бит атрибута равен 0, то обычно ищутся только файлы. Если в бите атрибута установлено значение метки тома, то будут возвращены только метки тома (если они есть). Любой другой атрибут или комбинация атрибутов (скрытый, системный и каталог) приводят к нахождению этих файлов, а также всех файлов без специального атрибута.

Пример. Найти на дисководе C в каталоге \MYDIR первый файл типа .COM:

```
fname   db      'C:\MYDIR\*.COM',0

dbuff   db      43 dup (0)      ; Получает результаты поиска
        .
        .
        .
                                ; Установка адреса DTA
        mov     ah,1ah          ; Номер функции
        mov     dx,seg dbuff    ; Адрес буфера результата
        mov     ds,dx
        mov     dx,offset dbuff
        int     21h             ; Переход в MS-DOS

                                ; Поиск первого соответствия
        mov     ah,4eh          ; Номер функции
        mov     cx,0            ; Без специального атрибута
        mov     dx,seg fname    ; Адрес имени файла
        mov     ds,dx
        mov     dx,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если нет соответствий
        .
        .
        .
```

Int 21H **[2.0]**
Функция 4FH(79)
Найти следующий файл

Предполагая успешным предшествующее обращение к прерыванию Int 21H с функцией 4EH, ищет на заданном или действующем по умолчанию дисководе в заданном или действующем по умолчанию каталоге следующий файл, который соответствует указанной ранее спецификации файла.

При вызове. AH = 4FH
Предполагается, что DTA – это рабочий буфер, использовавшийся предыдущим обращением к прерыванию Int21H с функциями 4EH или 4FH

При возврате. Если функция успешно выполнена (найден соответствующий файл)
Флаг переноса = сброшен
и результаты поиска возвращены в область передачи текущего диска так, как это описано для прерывания Int 21H с функцией 4EH
Если функция не выполнена
(соответствующего файла больше нет)
Флаг переноса = установлен
AX = код ошибки

Замечания. Использование данного вызова предполагает, что исходная спецификация файла, переданная при прерывании Int 21H с функцией 4EH, содержала один или более шаблонов групповых операций «*» или «?».

В случае вызова данной функции область передачи текущего диска (DTA) должна содержать информацию, полученную из предыдущего успешного обращения к прерыванию Int 21H с функцией 4EH или 4FH.

Пример. Продолжая операцию поиска в примере для прерывания Int 21H с функцией

4EH, найти (если он есть) на дисководе C в каталоге \MYDIR следующий файл типа .COM:

```
fname   db      'C:\MYDIR\*.COM',0

dbuff   db      43 dup (0)      ; Получает результаты поиска
        .
        .
        .
                                ; Поиск следующего соответствия
        mov     ah,4fh          ; Номер функции
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если список файлов исчерпан
        .
        .
        .
```

Int 21H
Функция 50H(80)
Зарезервирована

Int 21H
Функция 51H(81)
Зарезервирована

Int 21H
Функция 52H(82)
Зарезервирована

Int 21H
Функция 53H(83)
Зарезервирована

Int 21H **[2.0]**
Функция 54H(84)
Получить флаг проверки

Получает текущее значение системного флага проверки (чтением после записи).
При вызове. AH = 54H
При возврате. AL = текущее значение флага проверки
00H, если проверка выключена
01H, если проверка включена
Замечания. Так как проверка чтением после записи замедляет дисковые операции, то по умолчанию состояние системного флага проверки – *сброшен.*

При обращении к прерыванию Int 21H с функцией 2EH или с помощью команд системы MS-DOS VERIFY ON и VERIFY OFF можно изменить состояние системного флага проверки.
Пример. Получить состояние системного флага проверки:

```
        mov     ah,54h          ; Номер функции
```

```
        int     21h             ; Переход в MS-DOS
        cmp     al,,01h         ; Проверить состояние флага проверки
        je      label1          ; Перейти, флаг проверки установлен,
                                ; иначе полагаем, что флаг сброшен
```

Int 21H **[2.0]**
Функция 56H(86)
Переименовать файл

Изменяет имя файла и/или перемещает его запись из данного каталога в другой на том же диске. В среде системы MS-DOS версии 3.0 и старше данная функция может использоваться также и для изменения имен каталогов.

При вызове. AH = 56H
DS:DX = сегмент : относительный_адрес текущего пути, заданного в коде ASCIIZ
ES:DI = сегмент : относительный_адрес нового пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Функция не выполняется, если:
- не существует какой-либо элемент пути;
- уже существует файл с новым описанием пути;
- спецификация текущего пути содержит дисковод, отличный от указанного в новом описании пути;
- файл перемещается в корневой каталог, который переполнен;
- [3.0+] программа выполняется в сети, а пользователь имеет недостаточные права доступа либо к существующему файлу, либо к новому каталогу.

Не допускается применения шаблонов групповых операций «*» и «?» в описании ни текущего, ни нового пути.

Пример. Переименовать файл MYFILE.DAT, находящийся на дисководе C в каталоге \MYDIR, назвав его MYTEXT.DAT. Одновременно перенести файл в каталог \SYSTEM, расположенный на том же дисководе:

```
oldname db      'C:\MYDIR\MYFILE.DAT',0

newname db      'C:\SYSTEM\MYTEXT.DAT',0
        .
        .
        .
        mov     ah,56h          ; Номер функции
        mov     dx,seg oldname  ; Адрес старого имени файла
        mov     ds,dx
        mov     dx,offset oldname
        mov     di,seg newname  ; Адрес нового имени файла
        mov     es,di
        mov     di,offset newname
        int     21h             ; Переход в MS-DOS
```

```
        jc      error           ; Перейти, если операция
                                ; переименования не выполнена
        .
        .
        .
```

Int 21H **[2.0]**
Функция 57H(87)
Получить или установить дату и время создания файла

Получает или изменяет в записи каталога о файле время и дату его создания.

При вызове. В случае получения даты и времени
AH = 57H
AL = 00H
BX = дескриптор
В случае установки даты и времени
AH = 57H
AL = 01H
BX = дескриптор
CX = время
биты 00H–04H = 2-секундные приращения (0–29)
биты 05H–0AH = минуты (0–59)
биты 0BH–0FH = часы (0–23)
DX = дата
биты 00H–04H = день (1–31)
биты 05H–08H = месяц (1–12)
биты 09H–0FH = год (относительно 1980 г.)

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
и, если при вызове AL = 00H, то
CX = время
DX = дата
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Файл должен быть ранее открыт или создан с помощью вызова прерывания Int 21H с функциями 3CH, 3DH, 5AH, 5BH и 6CH.

Дата и время создания файла не показываются в листинге каталога, если все 16 бит, отводимые под них, сброшены.

Время и дата, установленные с помощью этой функции, будут действительны, даже если впоследствии файл модифицируется до закрытия дескриптора.

Пример. Получить дату создания или последней модификации файла MYFILE.DAT, а затем распаковать дату на составляющие ее части в переменные month, day и year:

```
fname   db      'MYFILE.DAT',0

month   dw      0
day     dw      0
year    dw      0
        .
        .
        .
                                ; Первая операция открытия файла
```

```
        mov     ah,3dh          ; Номер функции
        mov     al,0            ; Режим только для чтения
        mov     dx,seg fname    ; Адрес имени файла
        mov     ds,dx
        mov     dx,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция открытия
                                ; не выполнена

                                ; Получение даты и времени создания файла
        mov     bx,ax           ; Скопировать дескриптор в BX
        mov     ah,57h          ; Номер функции
        mov     al,0            ; 0 = получить подфункцию
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция не выполнена

        mov     day,dx          ; Декомпозиция даты
        and     day,01fh        ; Изолировать день
        mov     cl,5
        shr     dx,cl
        mov     month,dx        ; Изолировать месяц
        and     month,0fh
        mov     cl,4
        shr     dx,cl           ; Изолировать год
        and     dx,03fh         ; относительно 1980
        add     dx,1980         ; Поправка на текущий год
        mov     year,dx         ; Сохранить год

                                ; Операция закрытия файла,
                                ; дескриптор находится в BX
        mov     ah,3eh          ; Номер функции
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция закрытия
                                ; не выполнена
        .
        .
        .
```

Int 21H **[3.0]**
Функция 58H(88)
Получить или установить стратегию выделения

Получает или изменяет стратегию текущей версии MS-DOS для выделения блоков памяти.

При вызове. В случае получения стратегии
AH = 58H
AL = 00H
В случае установки стратегии
AH = 58H
AL = 01H
BX = требуемая стратегия
00H = поиск от начала
01H = оптимальный поиск
02H = поиск от конца

При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
и, если при вызове AL = 00H, то

AX = текущая стратегия
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Стратегии выделения памяти:

– **поиск от начала:** MS-DOS просматривает свободные блоки памяти от младших адресов к старшим и назначает первый блок, достаточный по величине для удовлетворения запроса;

– **оптимальный поиск:** MS-DOS просматривает все свободные блоки памяти и назначает наименьший блок, удовлетворяющий запросу, независимо от его местоположения;

– **поиск от конца:** MS-DOS просматривает свободные блоки памяти от старших адресов к младшим и назначает старший блок, достаточный по величине для удовлетворения запроса.

По умолчанию стратегия MS-DOS выделения памяти – поиск от начала (код 0).

Пример. Сохранить код, указывающий текущую стратегию выделения памяти, в переменной start, затем изменить системную стратегию выделения памяти на оптимальный поиск:

```
strat   dw      0               ; Предыдущий код стратегии
        .
        .
        .
                                ; Получение кода текущей стратегии
        mov     ah,58h          ; Номер функции
        mov     al,0            ; 0 = получить код стратегии
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        mov     start,ax        ; Сохранить код стратегии

                                ; Установить новый код стратегии
        mov     ah,58h          ; Номер функции
        mov     al,1            ; 1 = установить стратегию
        mov     bx,1            ; 1 = наилучшая обработка
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        .
        .
        .
```

Int 21H **[3.0]**
Функция 59H(89)
Получить расширенную информацию об ошибке

Получает после предыдущего неудачного обращения к прерыванию Int 21H детальную информацию об ошибке, включая рекомендуемое действие по исправлению.

При вызове. AH = 59H
BX = 00H

При возврате. AX = расширенный код ошибки

01H	неверный номер функции
02H	файл не найден
03H	путь не найден
04H	открыто слишком много файлов
05H	доступ запрещен

06H	недопустимый дескриптор
07H	уничтожены блоки управления памятью
08H	не хватает памяти
09H	недопустимый адрес блока памяти
0AH(10)	недопустимое окружение
0BH(11)	недопустимый формат
0CH(12)	недопустимый код доступа
0DH(13)	недопустимые данные
0EH(14)	неизвестное устройство
0FH(15)	недопустимый дисковод
10H(16)	попытка удалить текущий каталог
11H(17)	не то же устройство
12H(18)	нет больше файлов
13H(19)	попытка записи на защищенный диск
14H(20)	неизвестное устройство
15H(21)	дисковод не готов
16H(22)	неизвестная команда
17H(23)	ошибка данных (CRC)
18H(24)	неверная длина структуры запроса
19H(25)	ошибка позиционирования
1AH(26)	неизвестный тип носителя
1BH(27)	сектор не найден
1CH(28)	в принтере нет бумаги
1DH(29)	ошибка записи
1EH(30)	ошибка чтения
1FH(31)	общая ошибка
20H(32)	нарушение разделения
21H(33)	нарушение запирания
22H(34)	недопустимая смена дискеты
23H(35)	FCB отсутствует
24H(36)	разделяемый буфер переполнен
25H–31H	(37–49) зарезервированы
32H(50)	неподдерживаемый сетевой запрос
33H(51)	удаленная машина не слушает
34H(52)	дублирование имени в сети
35H(53)	сетевое имя не найдено
36H(54)	сеть занята
37H(55)	устройство больше не существует в сети
38H(56)	превышен предел команд NetBIOS
39H(57)	аппаратная ошибка сетевого адаптера
3AH(58)	неправильный отклик сети
3BH(59)	неожиданная ошибка сети
3CH(60)	несовместимость удаленного адаптера
3DH(61)	очередь к принтеру заполнена
3EH(62)	недостаточно места для файла принтера
3FH(63)	файл принтера удален
40H(64)	сетевое имя удалено
41H(65)	доступ к сети невозможен
42H(66)	неправильный тип сетевого устройства
43H(67)	сетевое имя не найдено
44H(68)	превышен предел сетевого имени

45H(69) превышен предел сетевого сеанса NetBIOS
46H(70) разделение файлов временно приостановлено
47H(71) сетевой запрос не принят
48H(72) приостановлено перенаправление принтера или диска
49H–4FH (73–79) Зарезервированы
50H(80) файл уже существует
51H(81) зарезервирован
52H(82) каталог не может быть создан
53H(83) отказ по прерыванию Int 24H (критическая ошибка)
54H(84) слишком много перенаправлений
55H(85) двойное перенаправление
56H(86) неверный пароль
57H(87) неверный параметр
58H(88) ошибка записи по сети
59H(89) в сети функция не поддерживается
5AH(90) требуемый компонент системы не установлен

BH = класс ошибки
01H, если вне ресурса (такого, как память или дескрипторы)
02H, если не ошибка, но временная ситуация (например, запертый район в файле), которая, возможно, изменится
03H, если проблема санкционирования
04H, если внутренняя ошибка системного программного обеспечения
05H, если отказ аппаратуры
06H, если отказ системного программного обеспечения, а не ошибка активного процесса (например, потеря файлов конфигурации)
07H, если ошибка прикладной программы
08H, если файл или элемент данных не найден
09H, если файл или элемент данных недопустимого типа или формата
0AH(10), если файл или элемент данных заперт
0BH(11), если на дисководе неправильный диск, дефект на диске или проблема магнитного накопителя
0CH(12), если объект уже существует
0DH(13), неизвестная ошибка

BL = рекомендуемые действия
01H повторить разумное число раз, затем предложить пользователю выбрать либо завершение, либо игнорирование ошибки
02H повторить разумное число раз с задержкой между повторами, затем предложить пользователю выбрать либо завершение, либо игнорирование ошибки
03H получить от пользователя корректирующую информацию (обычно обусловлено неправильной спецификацией файла или дисковода)
04H прервать прикладную программу с уборкой (т. е. завершить программу самым строгим образом: освободить запирания, закрыть файлы и т. д.)
05H выполнить немедленный выход без уборки

06H игнорировать ошибку
07H повторить после вмешательства пользователя с целью удалить причину ошибки

CH = местоположение ошибки
01H неизвестно
02H блочно-ориентированное устройство (диск или эмулятор диска)
03H сеть
04H последовательное устройство
05H память

И для системы MS-DOS версий 3.0 и старше
ES:DI = метка тома (в коде ASCIIZ) диска для установки, если AX = 0022H (неправильная смена диска)

Замечания. Данную функцию можно вызывать после обращения к любой другой функции прерывания Int 21H, которая возвратила признак ошибки, с целью получить более детальную информацию о типе ошибки и рекомендации по ее устранению. Если предыдущее обращение к функции прерывания Int 21H не дало ошибку, то в регистр AX возвращается значение 0000H. Данную функцию можно также вызывать и во время работы обработчика критической ошибки (прерывание Int 24H).

Содержимое регистров CL, DX, SI, DI, BP, DS и ES разрушается данной функцией.

Отметим, что коды расширенной ошибки 13H–1FH (19–31) и 34(22H) в точности соответствуют кодам ошибки 0–0CH (0–12) и 0FH (15), возвращаемым прерыванием Int 24H.

Если вы хотите гарантировать совместимость снизу вверх, то вам не следует использовать в своей программе только заданные номера ошибок, так как в каждой версии системы MS-DOS добавляются новые коды ошибок.

Пример. С помощью блока управления файлами попытаться открыть файл с именем NOSUCH.DAT; если запрос на открытие не выполняется, получить код расширенной ошибки:

```
myfcb   db      0               ; Дисковод = действующий по умолчанию
        db      'NOSUSH  '      ; Имя файла, 8 символов
        db      'DAT'           ; Расширение, 3 символа
        db      25 dup (0)      ; Остаток FCB
        .
        .
        .
label1:                         ; Открытие файла
        mov     ah,0fh          ; Номер функции
        mov     dx,seg myfcb    ; Адрес FCB
        mov     ds,dx
        mov     dx,offset myfcb
        int     21h             ; Переход в MS-DOS
        or      al,al           ; Проверить состояние операции открытия
        jz      success         ; Перейти, если операция открытия выполнена

                                ; Получение расширенной
                                ; информации об ошибке,
                                ; если операция открытия не выполнена
        mov     ah,59h          ; Номер функции
        xor     bx,bx           ; Содержимое BX должно = 0
        int     21h             ; Переход в MS-DOS
        or      ax,ax           ; Повторить проверку ошибки
        jz      success         ; Перейти, если ошибки нет
```

```
cmp     bl,2            ; Необходимо повторить?
jle     label1          ; Да, перейти
jmp     error           ; Нет, отбросить
.
.
```

Int 21H [3.0]
Функция 5AH(90)
Создать временный файл

Создает файл с уникальным именем в текущем или заданном каталоге действующего по умолчанию или заданного дисковода, а также возвращает дескриптор, который может использоваться программой для последующего доступа к файлу. Образованное имя файла также возвращается в заданный программой буфер.

При вызове. AH = 5AH
CX = атрибут (биты могут комбинироваться)

Бит (ы)	**Описание** (если бит установлен)
0	только для чтения
1	скрытый
2	системный
3–4	зарезервированы (0)
5	атрибут архива
6–15	зарезервированы (0)

DS:DX = сегмент : относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = дескриптор
DS:DX = сегмент : относительный_адрес полного пути, заданного в коде ASCIIZ
Если функция не выполнена.
Флаг переноса = установлен
AX = код ошибки

Замечания. За описанием в кодах ASCIIZ пути, которым снабжается данная функция, должно следовать по крайней мере 13 дополнительных байтов пространства буфера. Система MS-DOS добавляет знак «\» в составленный путь, если это необходимо, а затем присоединяет заканчивающееся нулем имя файла, которое является функцией текущего времени.

Созданные с помощью данной функции файлы автоматически не удаляются в тех случаях, когда вызывающая программа завершает свою работу.

Функция не выполняется, если:
– отсутствует хотя бы один элемент пути;
– файл создается в корневом каталоге, который переполнен.

См. также прерывание Int 21H с функциями 3CH, 5BH и 6CH, которое предоставляет дополнительные возможности для создания файлов.

[3.0+] Если программа выполняется в сети, то файл создается и открывается для доступа в операциях чтения-записи в совместимом режиме разделения.

Пример. Создать на дисководе C в каталоге \TEMP временный файл с уникальным именем без специальных атрибутов. Заметим, что вы должны предусмотреть место для того, чтобы MS-DOS добавила образованное имя к указанному пути. Для удаления

временного файла до завершения работы вашей программы следует использовать полную спецификацию файла.

```
fname   db      'C:\TEMP\'      ; Путь к временному файлу
        db      13 dup (0)      ; Получает имя файла

fhandle dw      ?               ; Дескриптор файла
        .
        .
        .
        mov     ah,5ah          ; Номер функции
        mov     cx,0            ; Без специального атрибута
        mov     dx,seg fname    ; Адрес пути
        mov     ds,dx
        mov     dx,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция создания
                                ; не выполнена
        mov     fhandle,ax      ; Сохранить дескриптор файла
        .
        .
        .
```

Int 21H **[3.0]**
Функция 5BH(91)
Создать новый файл

Получив путь в коде ASCIIZ, создает файл в указанном или действующем по умолчанию каталоге указанного или действующего по умолчанию дисковода, а также возвращает дескриптор, который можно использовать в программе для последующего доступа к файлу. Функция не выполняется, если файл с тем же именем уже существует.

При вызове. AH = 5BH
CX = атрибут (биты могут комбинироваться)

Бит (ы)	**Описание** (если бит установлен)
0	только для чтения
1	скрытый
2	системный
3	метка тома
4	зарезервирован (0)
5	атрибут архива
6–15	зарезервированы (0)

DS:DX = сегмент : относительный адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = дескриптор
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Функция не выполняется, если:
– не существует хотя бы одного элемента пути;
– уже существует файл с идентичным описанием пути (т. е. с тем же именем файла и расширением и с тем же положением в структуре каталога);

– файл создается в корневом каталоге, который переполнен;

– [3.0+] программа выполняется в сети, а пользователь имеет недостаточные права доступа к каталогу, в котором будет находиться файл.

При создании файла обычно ему не присваивается никаких специальных атрибутов (0), и он открыт для операций как записи, так и чтения. С помощью прерывания Int 21H с функцией 43H атрибуты файла можно последовательно изменять.

См. также прерывание Int 21H с функциями 5CH, 5AH и 6CH, которые предоставляют альтернативный путь создания файлов.

Данную функцию можно использовать для реализации семафоров в сетевой или многозадачной среде. Если функция завершается успешно, то программа получает семафор. Для освобождения семафора программа просто удаляет файл.

Пример. Создать и открыть в каталоге \MYDIR дисковода C файл с именем MYFILE.DAT; MS-DOS возвращает ошибку, если в том же месте уже существует файл с тем же именем;

```
fname   db      'C:\MYDIR\MYFILE.DAT',0

fhandle dw      ?               ; Дескриптор файла
        .
        .
        .
        mov     ah,5bh          ; Номер функции
        xor     cx,cx           ; Без специальных атрибутов
        mov     dx,seg fname    ; Адрес пути
        mov     ds,dx
        mov     dx,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция создания
                                ; не выполнена
        mov     fhandle,ax      ; Сохранить дескриптор файла
        .       .
        .
        .
```

Int 21H [3.0]

Функция 5CH(92)

Запереть или отпереть район файла

Запирает или отпирает определенный район файла. Данная функция недоступна, если не загружен модуль разделения файлов (SHARE.EXE).

При вызове. AH = 5CH
AL = 00H при запирании района
01H при отпирании района
BX = дескриптор
CX = старшая часть относительного адреса района
DX = младшая часть относительного адреса района
SI = старшая часть длины района
DI = младшая часть длины района

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Данная функция полезна для синхронизации файлов и записей в многозадачной среде или сети. Доступ к файлу в целом управляется параметрами, опре-

деляющими атрибуты и разделение файла, которые передаются при вызовах операций открытия или создания, а также через атрибуты файла, которые хранятся в соответствующей записи каталога.

Начальное положение запираемой или отпираемой части в файле задается целым положительным числом с двойной точностью, представляющим относительный адрес байта в файле. Длина запираемого или отпираемого района задается подобным же образом в виде целого положительного числа с двойной точностью.

После каждого вызова операции запирания района файла должен следовать вызов операции отпирания с точно таким же смещением в файле и такой же длиной.

Запирание за текущим концом файла не является ошибкой.

Дубликаты дескрипторов, созданные с помощью прерывания Int 21H с функцией 45H, как и дескрипторы, перенаправленные на файл с помощью прерывания Int 21H с функцией 46H, обеспечивают доступ к запертым районам внутри одного процесса.

Программы, которые загружаются через вызов функции EXEC (с помощью прерывания Int 21H с функцией 4BH), наследуют дескрипторы родительского процесса, но без активных запираний.

Если процесс завершается без освобождения активных запираний файла, то результат не определен. Кроме того, программы, использующие данную функцию, должны устанавливать собственные обработчики прерываний Int 23H и Int 24H с тем, чтобы избежать их неожиданного завершения.

Пример. Пусть ранее был открыт файл и его дескриптор сохранен в переменной fhandle. Запереть в файле район размером 4096 байт, начало которого смещено от начала файла на 32 768 байт, так чтобы другие программы доступа к нему не имели:

```
fhandle dw      ?               ; Дескриптор файла
        .
        .
        .
        mov     ah,5ch          ; Номер функции
        mov     al,0            ; Подфункция 0 = запирание
        mov     bx,fhandle      ; Дескриптор файла
        mov     cx,0            ; Старшая часть относительного адреса
        mov     dx,32768        ; Младшая часть относительного адреса
        mov     si,0            ; Старшая часть длины
        mov     di,4096         ; Младшая часть длины
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция
                                ; запирания не выполнена
```

Int 21H
Функция 5DH(93)
Зарезервирована

Int 21H [3.1]
Функция 5EH(94), подфункция 00H
Получить имя машины

Возвращает адрес ASCIIZ-строки (заканчивающейся нулем), идентифицирующей локальный компьютер. Обращение к этой функции возможно, только когда работает Microsoft Networks.

При вызове. AH = 5EH
AL = 00H
DS:DX = сегмент : относительный_адрес буфера,
хранящего строку
При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
CH = 00H, если имя не определено
00H, если имя определено
CL = номер имени netBIOS (если CH ≠ 0)
DS:DX = сегмент : относительный_адрес идентификатора
(если CH ≠ 0)
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Идентификатор компьютера – это строка из 15 байт, к которой добавлены пробелы с нулевым (00H) байтом в конце.

Действие вызова этой функции непредсказуемо, если не загружен модуль поддержки разделения файла.

Пример. Записать в буфер mname имя локального компьютера:

```
mname   db      16 dup (?)
        .
        .
        .
        mov     ax,5e00h        ; Функция и подфункция
        mov     dx,seg mname    ; Адрес буфера
        mov     ds,dx
        mov     dx,offset mname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена

        or      ch,ch           ; Удостоверимся, что имя существует
        jz      error           ; Перейти, если имя не определено
        .
        .
        .
```

Int 21H **[3.1]**
Функция 5EH(94), подфункция 02H
Задать строку установки принтера

Задает строку, которая будет передаваться перед всеми файлами, направляемыми на конкретный принтер сети, давая возможность пользователям с разных узлов сети задавать индивидуальные режимы работы на одном и том же принтере. Обращение к данной функции возможно, только если работает Microsoft Networks.

При вызове. AH = 5EH
AL = 02H
BX = индекс таблицы перенаправления
CX = длина строки установки
DS:SI = сегмент : относительный_адрес строки установки
При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен

AX = код ошибки

Замечания. Индекс таблицы перенаправления, передаваемый в регистр BX, получен с помощью функции 5FH, подфункции 02H (Получить запись таблицы перенаправления).

См. также функцию 5EH, подфункцию 03H, с помощью которой можно получить строку существующей установки конкретного принтера сети.

Пример. Инициализировать строку установки для принтера, указанного индексом 2 таблицы перенаправления, так чтобы устройство устанавливалось в режим жирного шрифта до печати файла, затребованного этим узлом сети.

```
setup   db      01bh,045h       ; Устанавливает режим жирного шрифта
        .
        .
        .
        mov     ax,5e02h        ; Функция и подфункция
        mov     bx,2            ; Индекс 2 таблицы перенаправления
        mov     cx,2            ; Длина строки установки
        mov     si,seg setup    ; Адрес строки установки
        mov     ds,si
        mov     si,offset setup
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        .
        .
        .
```

Int 21H **[3.1]**

Функция 5EH(94), подфункция 03H

Получить строку установки принтера

Получает строку установки конкретного принтера сети. Обращение к этой функции доступно, только если работает Microsoft Networks.

При вызове. AH = 5EH
AL = 03H
BX = индекс таблицы перенаправления
ES:DI = сегмент : относительный_адрес буфера, хранящего строку установки

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
CX = длина строки установки принтера
ES:DI = адрес буфера, содержащего строку установки
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Индекс таблицы перенаправления, передаваемый в регистр BX, получен с помощью функции 5FH, подфункции 02H (Получить запись таблицы перенаправления).

См. также прерывание Int 21H с функцией 5EH, подфункцией 02H, с помощью которой можно задать строку установки принтера сети.

Пример. Получить строку установки для этого узла сети, связанного с принтером, указанным индексом 2 таблицы перенаправления:

```
setup   db      64 dup (?)      ; Получает строку установки
```

```
.
.
.
mov     ax,5e03h        ; Функция и подфункция \
mov     bx,2            ; Индекс 2 таблицы перенаправления
mov     di,seg setup    ; Адрес буфера
mov     es,di
mov     di,offset setup
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если функция не выполнена
.
.
.
```

Int 21H [3.1]

Функция 5FH(95), подфункция 02H

Получить запись из таблицы перенаправления

Организует просмотр системной таблицы перенаправления, которая связывает логические имена с сетевыми файлами, каталогами или принтерами. Обращение к данной функции возможно, только если работает Microsoft Networks и загружен модуль разделения файла (SHARE.EXE).

При вызове. AH = 5FH
AL = 02H
BX = индекс таблицы перенаправления
DS:SI = сегмент : относительный_адрес буфера
длиной 128 байт для хранения сетевого имени

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
BX = флаг состояния устройства
Бит 0 = 0, если допустимое устройство
= 1, если недопустимое устройство
BL = тип устройства
03H, если принтер
04H, если устройство
CX = значение сохраненного параметра
DX = содержимое разрушается
BP = содержимое разрушается
DS:SI = сегмент : относительный_адрес имени локального устройства, заданного в коде ASCIIZ
ES:DI = сегмент : относительный_адрес сетевого имени, заданного в коде ASCIIZ
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. Параметр, возвращаемый в CX, определяется значением, ранее переданным через регистр CX в MS-DOS с помощью прерывания Int 21H с функцией 5FH, подфункцией 03H (Перенаправить устройство). Параметр представляет данные, значимые для прикладной программы, которая сохраняет и восстанавливает их, и незначимые для MS-DOS.

Пример. Получить локальные и сетевые имена устройства, заданного первой записью в таблице перенаправления:

```
local   db      64 dup (?)      ; Получает локальное имя устройства

network db      128 dup (?)     ; Получает сетевое имя
        .
        .
        .
        mov     ax,5f02h        ; Функция и подфункция
        mov     bx,0            ; Запись 0 таблицы перенаправления
        mov     si,seg local    ; Адрес буфера локального имени
        mov     ds,si
        mov     si,offset local
        mov     di,seg network  ; Адрес буфера сетевого имени
        mov     es,di
        mov     di,offset network
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если вызов не выполнен

        or      bh,bh           ; Проверить состояние устройства
        jnz     error           ; Перейти, если это
                                ; недопустимое устройство
```

Int 21H [3.1]
Функция 5FH(95), подфункция 03H
Перенаправить устройство

Устанавливает перенаправление через сеть, соединяя локальное имя устройства с сетевым именем. Обращение к этой функции возможно только при условии, что работает Microsoft Networks и загружен модуль разделения файлов (SHARE.EXE).

При вызове. AH = 5FH
AL = 03H
BL = тип устройства
03H, если принтер
04H, если дисковод
CX = параметр для сохранения вызывающего оператора
DS:SI = сегмент : относительный_адрес локального имени устройства, заданного в коде ASCIIZ
ES:DI = сегмент : относительный_адрес сетевого имени, заданного в коде ASCIIZ и следующего за ним ASCIIZ-паролем

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Локальным именем могут быть обозначение дисковода (буква с двоеточием за ней, например «D:»), имя принтера и нулевая строка. Имя принтера должно быть одним из следующих: PRN, LRT1, LPT2 и LPT3. Если перед паролем используется нулевая строка, то система MS-DOS пытается предоставить доступ к каталогу сети с заданным паролем.

Параметр, передаваемый в CX, можно отыскать с помощью последующих обращений к прерыванию Int 21H с функцией 5FH, подфункцией 02H. Параметр представляет

данные, значимые для прикладной программы, которая сохраняет и восстанавливает их, и незначимые для MS-DOS.
Пример. Перенаправить, используя пароль FRED, локальный дисковод E в каталог \FORTH на сервере с именем LMI:

```
locname db      'E:',0          ; Локальный дисковод

netname db      '\\LMI\FORTH',0
        db      'FRED',0
        .
        .
        .
        mov     ax,5f03h        ; Функция и подфункция
        mov     bl,4            ; Код 4 = дисковод диска
        mov     si,seg locname  ; Адрес локального имени
        mov     ds,si
        mov     si,offset locname
        mov     di,seg netname  ; Адрес сетевого имени
        mov     es,di
        mov     di,offset netname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция перенаправления
                                ; не выполнена
```

Int 21H [3.1]
Функция 5FH(95), подфункция 04H
Отменить перенаправление устройства

Отменяет предыдущий запрос на перенаправление путем удаления связи локального имени устройства с сетевым именем. Вызов данной функции возможен, только если работает Microsoft Networks и загружен модуль разделения файлов (SHARE.EXE).
При вызове. AH = 5FH
AL = 04H
DS:SI = сегмент : относительный адрес локального имени устройства, заданного в коде ASCIIZ
При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки
Замечание. Передаваемым именам могут быть обозначение дисковода (буква с последующим двоеточием, например «D:»), имя принтера или строка символов, которая начинается двумя знаками «\\». Имя принтера должно быть одним из следующих: PRN, LPT1, LPT2 и LPT3. Если используется строка с двумя знаками «\\» в начале, то связь между локальной машиной и сетевым каталогом завершается.
Пример. Отменить перенаправление локального дисковода E на сетевой сервер:

```
locname db      'E:',0
        .
        .
        .
```

```
        mov     ax,5f04h        ; Функция и подфункция
        mov     si,seg locname  ; Адрес локального имени
        mov     ds,si
        mov     si,offset locname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция
                                ; отмены не выполнена
        .
        .
        .
```

Int 21H
Функция 60H(96)
Зарезервирована

Int 21H
Функция 61H(97)
Зарезервирована

Int 21H **[3.0]**
Функция 62H(98)
Получить адрес PSP

Получает адрес сегмента (в параграфах) префикса программного сегмента (PSP) для выполняющейся в текущий момент программы.
При вызове. AH = 62H
При возврате. BX = адрес сегмента префикса программного сегмента
Замечания. Прежде чем программа получает управление от MS-DOS, ее префикс программного сегмента заполняется определенной существенной информацией, куда входят:

- адрес сегмента блока окружения программы;
- командная строка, первоначально введенная пользователем;
- исходное содержимое векторов обработчиков критической ошибки, Ctrl-C и завершения;
- старший адрес доступной части ОЗУ.

Обычно адрес сегмента PSP передается в программу через регистры DS и ES в момент, когда MS-DOS передает управление программе. Данная функция позволяет программе удобным образом восстанавливать адрес PSP во время выполнения в любой точке, без необходимости сохранять этот адрес при входе в программу.
Пример. Получить базу сегмента префикса программного сегмента, а затем скопировать из PSP в локальный буфер buff хвост команды:

```
ctail   equ     080H            ; Относительный адрес в PSP, хвост команды

buff    db      80 dup (?)      ; Копировать хвост команды
        .
        .
        .
                                ; Получение адреса PSP
        mov     ah,62H          ; Номер функции
        int     21h             ; Переход в MS-DOS
                                ; Копирование хвоста команды
        mov     ds,bx           ; Сегмента PSP в DS
        mov     si,offset ctail ; Относительный адрес хвоста команды
```

```
mov     di,seg buff     ; Адрес локального буфера
mov     es,di
mov     di,offset buff
mov     cl,[si]         ; Длина хвоста команды
inc     cl              ;Инкремент счетчика байтов
xor     ch,ch
cld
rep movsb               ; Копировать в локальный буфер
.
.
.
```

Int 21H **[2.25 только]**

Функция 63H(99)

Получить таблицу лидирующих байтов для DBCS

Получает адрес системной таблицы допустимой области лидирующих байтов для двухбайтного набора символов (double-byte character set, DBCS) и, кроме того, устанавливает или получает флаг временной консоли. Прерывание Int 21H с функцией 63H доступно только в среде MS-DOS версий 2.25; оно не поддерживается в среде MS-DOS версий 3.0 и старше.

При вызове. AH = 63H
AL = подфункция
00H в случае получения таблицы лидирующих байтов
01H в случае установки или сброса флага временной консоли
02H в случае получения значения флага временной консоли
Если AL = 01H
DL = 00H в случае сброса флага временной консоли
01H в случае установки флага временной консоли

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
и, если при вызове AL = 00H, то
DS:SI = сегмент : относительный_адрес таблицы лидирующих байтов DBCS
или, если при вызове AL = 02H, то
DL = значение флага временной консоли
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Таблица лидирующих байтов DBCS состоит из переменного числа двухбайтных записей, в конце которой помещены два нулевых (00H) байта. Каждая пара определяет начальное и конечное значения области лидирующих байтов. Допустимые значения лидирующих байтов всегда лежат в области 80–0FFH.

Записи в таблице лидирующих байтов должны располагатъся в возрастающем порядке. Если в данной системе не определено ни одного лидирующего байта, то таблица состоит только из двухбайтного нуля.

Если флаг временной консоли установлен, то прерывания Int 21H с функциями 07H (Нефильтрованный ввод без эхо) и 0BH (Проверить состояние ввода) будут поддерживать временные символы.

В отличие от большинства других сервисных функций системы MS-DOS вызов данной функции не обязательно сохраняет регистры, за исключением SS:SP.

[4.0] С помощью прерывания Int 21H с функцией 65H также можно получить адрес таблицы лидирующих байтов DBCS.

Int 21H
Функция 64H(100)
Зарезервирована

Int 21H **[3.3]**
Функция 65H(101)
Получить расширенную информацию о национальном формате

Получает информацию о заданной стране и/или кодовой странице
При вызове: AH = 65H
AL = подфункция
01H = получить расширенную информацию о национальном формате
02H = получить указатель таблицы отображения символов верхнего регистра
04H = получить указатель таблицы отображения символов верхнего регистра имени файла
06H = получить указатель таблицы соответствия
07H = получить указатель вектора таблицы лидирующих байтов (DBCS)
(MS-DOS версий 4.0 и старше)
BX = требуемая кодовая страница (— 1 = активное устройство CON)
CX = длина буфера для получения информации (должна быть ⩾ 5)
DX = ID страны (— 1 = по умолчанию)
ES:DI = адрес буфера для получения информации
При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
и запрашиваемые данные помещены в буфер вызывающей программы
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Информация, возвращаемая данной функцией,— это надмножество информации, возвращаемой с помощью прерывания Int 21H с функцией 38H.

Данная функция может не выполняться в тех случаях, когда недопустимы либо код страны, либо номер кодовой страницы или если кодовая страница не соответствует коду страны.

Функция не выполняется, если заданная длина буфера меньше 5 байт. В том случае, когда длина буфера, предназначенного для хранения информации, по крайней мере 5 байт, но значительно меньше, чем объем запрашиваемой информации, то данные усекаются без возврата кода ошибки.

Формат данных, возвращаемых подфункцией 01H:

Байт (ы)	**Описание**
00H	идентификатор (ID) информации (1)
01H–02H	длина последующего буфера
03H–04H	идентификатор страны
05–06H	номер кодовой страницы
07H–08H	формат даты
	0 = США месяц, день, год

1 = Европа день, месяц, год
2 = Япония год, месяц, день
09H–0DH код ASCIIZ идентификатора валюты
0EH–0FH код ASCIIZ разделителя тысяч
10H–11H код ASCIIZ десятичной точки
12H–13H код ASCIIZ разделителя даты
14H–15H код ASCIIZ разделителя времени
16H флаги форматов валюты
бит 0 = 0, если идентификатор валюты предшествует значению
= 1, если идентификатор валюты следует за значением
бит 1 = 0, если пробел между значением и идентификатором валюты отсутствует
= 1, если между значением и идентификатором валюты один пробел
бит 2 = 0, если идентификатор валюты и десятичная точка разделены
= 1, если идентификатор валюты заменяет десятичную точку
17H число цифр после десятичной точки в представлении валюты
18H формат времени
бит 0 = 0, если 12-часовое представление
= 1, если 24-часовое представление
19H–1CH адрес вызова процедуры отображения символов верхнего регистра
1DH–1EH код ASCIIZ разделителя данных
1FH–28H зарезервированы

Формат данных, возвращаемых подфункциями 02H, 04H, 06H и 07H:

Байт (ы)	Описание
00H	идентификатор (ID) информации (2, 4 или 6)
01H–05H	двухсловный указатель таблицы

Максимальная длина таблиц отображения символов верхнего регистра и символов верхнего регистра имени файла – 130 байт. Первые два байта содержат размер таблицы; последующие байты содержат эквиваленты символов верхнего регистра, если они есть, для символьных кодов 80H–FFH. Основное назначение этих таблиц – отображение акцентированных или иным способом модифицированных гласных в эквивалентные им простые гласные.

Текст, оттранслированный с помощью этой таблицы, может передаваться на устройство, которое не поддерживает графический набор символов IBM, или использоваться для создания имен файлов, не требующих при записи специальной конфигурации клавиатуры.

Максимальная длина таблицы соответствия – 258 байт. Первые два байта содержат длину таблицы, в последующих содержатся величины, которые будут использоваться для соответствующих символьных кодов (0–FFH) во время операции упорядочивания. Данная таблица отображает ASCII-символы верхнего и нижнего регистров в одни и те же коды соответствия, так чтобы упорядочивание осуществлялось независимо от регистра, а кроме того, она отображает акцентированные гласные в эквивалентные им простые гласные.

[4.0+] Подфункция 07H возвращает указатель таблицы переменной длины, которая определяет области лидирующих байтов для двухбайтного набора символов (DBCS). Таблица заканчивается парой нулевых байтов, если только она не должна усекаться в целях подгонки под размер буфера, и имеет следующий формат:

```
        dw      length
        db      start1,end1
        db      start2,end2
        .
        .
        .
        db      0,0
```

Например:

```
        dw      4
        db      81h,9fh
        db      0e0h,0fch
        db      0,0
```

В некоторых случаях усеченная таблица трансляции может предоставляться программе системой MS-DOS. В прикладных программах всегда следует проверять длину в начале таблицы, чтобы удостовериться в том, что она содержит код трансляции для конкретного интересующего нас символа.

Примеры. Получить расширенную информацию о национальном формате страны, действующем по умолчанию, и кодовую страницу 437:

```
buffer  db      41 dup (0)      ; Получает информацию о
                                ; национальном формате
        .
        .
        .
        mov     ax,6501h        ; Функция и подфункция
        mov     bx,437          ; Кодовая страница
        mov     cx,41           ; Длина буфера
        mov     dx,-1           ; Страна по умолчанию
        mov     di,seg buffer   ; Адрес буфера
        mov     es,di
        mov     di,offset buffer
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        .
        .
        .
```

В этом случае MS-DOS помещает в буфер следующую расширенную информацию о национальном формате:

```
buffer  db      1               ; ID информации
        dw      38              ; Длина последующего буфера
        dw      1               ; Идентификатор страны (США)
        dw      437             ; Номер кодовой страницы
        dw      0               ; Формат даты
        db      '$',0,0,0,0     ; Идентификатор валюты
        db      ',',0           ; Разделитель тысяч
        db      '.',0           ; Десятичный разделитель
        db      '-',0           ; Разделитель даты
        db      ':',0           ; Разделитель времени
        db      0               ; Флаги форматов валюты
        db      2               ; Цифры в валюте
        db      0               ; Формат времени
```

```
        dd      026ah:176ch     ; Точка входа процедуры отображения
                                ; верхнего регистра клавиатуры
        db      ',',0           ; Разделитель таблицы данных
        db      10 dup (0)      ; Зарезервировано
```

Получить указатель таблицы отображения верхнего регистра, соответствующей действующему по умолчанию формату страны, и кодовую страницу 437:

```
buffer  db      5 dup (0)       ; Получает информацию об указателе
        .
        .
        .
        mov     ax,6502h        ; Номер функции
        mov     bx,437          ; Кодовая страница
        mov     cx,5            ; Длина буфера
        mov     dx,-1           ; Страна по умолчанию
        mov     di,seg buffer   ; Адрес буфера
        mov     es,di
        mov     di,offset buffer
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если функция не выполнена
        .
        .
        .
```

В этом случае MS-DOS помещает в буфер следующие величины:

```
buffer  db      2               ; ID информации
        dw      0204h           ; Относительный адрес таблицы отображения
                                ; верхнего регистра клавиатуры
        dw      1140h           ; Сегмент таблицы отображения
                                ; верхнего регистра клавиатуры
```

Расположенная по адресам 1140:0204H таблица содержит следующие данные:

```
              0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
1140:0200                    80 00 80 9A 45 41 8E 41 8F 80 45 45      ....EA.A..EE
1140:0210  45 49 49 49 8E 8F 90 92 92 4F 99 4F 55 55 59 99  EIII.....O.OUUY.
1140:0220  9A 9B 9C 9D 9E 9F 41 49 4F 55 A5 A5 A6 A7 A8 A9  ......AIOU......
1140:0230  AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7 B8 B9  ................
1140:0240  BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7 C8 C9  ................
1140:0250  CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7 D8 D9  ................
1140:0260  DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7 E8 E9  ................
1140:0270  EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7 F8 F9  ................
1140:0280  FA FB FC FD FE FF                                ......
```

Int 21H [3.3]
Функция 66H(102)
Получить или установить кодовую страницу

Получает или выбирает текущую кодовую страницу
При вызове. AH = 66H
AL = подфункция
01H = получить кодовую страницу
02H = выбрать кодовую страницу
BX = выбираемая кодовая страница при AL = 02H
При возврате. Если функция успешно выполнена

Флаг переноса = сброшен
и, если при вызове AL = 01H
BX = активная кодовая страница
DX = кодовая страница по умолчанию
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. В тех случаях, когда используется подфункция «Выбрать кодовую страницу», система MS-DOS получает новую кодовую страницу из файла COUNTRY.SYS. Устройство должно заранее подготавливаться для переключения кодовой страницы с помощью директивы DEVICE = в файле CONFIF.SYS и команд NLSFUNC и MODE CP PREPARE (обычно расположенных в файле AUTOEXEC.BAT)

Пример. Сделать так, чтобы активная кодовая страница была такой же, как действующая по умолчанию системная кодовая страница, т.е. восстановить ту кодовую страницу, которая была активной в момент первоначальной загрузки системы:

```
.
.
.
                        ; Получение текущей и действующей по
                        ; умолчанию кодовой страницы
mov     ax,6601h        ; Номер функции
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если функция не выполнена

                        ; Установка кодовой страницы
mov     bx,dx           ; Активная = действующая по умолчанию
mov     ax,6502h        ; Номер функции
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если функция не выполнена
.
.
.
```

Int 21H [3.3]
Функция 67H(103)
Установить число дескрипторов

Устанавливает максимальное число файлов или устройств, которые могут быть одновременно открыты с помощью дескрипторов в текущем процессе.

При вызове. AH = 67H
BX = желательное число дескрипторов

При возврате. Если функция выполнена успешно
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Вызов этой функции управляет размером таблицы, которая устанавливает соответствие номеров дескрипторов для текущего процесса и глобальной таблицы системы MS-DOS для всех открытых в системе файлов и устройств. Таблица, действующая по умолчанию, располагается в резервируемой области PSP процесса и достаточна для хранения 20 одновременно открытых дескрипторов.

Функция не выполняется, если запрашиваемое число дескрипторов больше 20

и в системе недостаточно свободной памяти для выделения нового блока, необходимого для хранения увеличенной таблицы.

Если число затребованных дескрипторов больше числа доступных записей в системной глобальной таблице для дескрипторов файлов и устройств (число записей определяется директивой FILES в файле CONFIG.SYS), то ошибка не возвращается. Однако последующая попытка открыть файл или устройство или создать новый файл не достигнет результата, если все записи в системной глобальной таблице файлов задействованы, даже при условии, что запрашивающий процесс не израсходовал все свои дескрипторы.

Пример. Установить для текущего процесса максимальное число дескрипторов равным 30, так чтобы процесс мог иметь до 30 одновременно открытых файлов или устройств. (Пять дескрипторов сразу же назначаются стандартным устройствам при запуске процесса.) Заметим, что для успешного открытия 30 файлов или устройств в одном процессе, необходимо также затребовать запись FILES = 30 (или большее значение) в файле CONFIG.SYS.

```
.
.
.
mov     ah,67h          ; Номер функции
mov     bx,30           ; Максимальное число дескрипторов
int     21h             ; Переход в MS-DOS
jc      error           ; Перейти, если функция не выполнена
.
.
.
```

Int 21H **[3.3]**
Функция 68H(104)
Сбросить в файл

Принудительно переносит (физически) все данные из внутренних буферов системы MS-DOS, связанных с заданным дескриптором, на устройство. Если дескриптор относится к файлу, а файл модифицировался, то в соответствующей записи каталога обновляются размер файла, а также дата и время его последней модификации.

При вызове. AH = 68H
BX = дескриптор

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Действие этой функции эквивалентно закрытию и повторному открытию файла, или дублированию с помощью прерывания Int 21H с функцией 45H дескриптора файла, а затем закрытию дубликата. Однако данная функция имеет преимущество, так как она будет выполняться и при недостаточном числе дескрипторов, и прикладная программа не рискует потерять управление файлом в многозадачных или сетевых средах.

Если данная функция запрашивается для дескриптора, связанного с символьно-ориентированным устройством, то возвращается признак успешного выполнения, хотя никаких действий не выполняется.

Пример. Пусть файл MYFILE.DAT ранее был открыт, причем дескриптор этого файла хранится в переменной fhandle. Функция «Сбросить в файл» вызывается для того,

чтобы гарантировать, что данные во внутренних буферах системы MS-DOS, связанные с дескриптором, переписаны на диск, а каталог и таблица размещения файлов модифицированы должным образом.

```
fname   db      'MYFILE.DAT',0  ; Строка ASCIIZ с именем файла
fhandle dw      ?               ; Дескриптор файла
        .
        .
        .
        mov     ah,68h          ; Номер функции
        mov     bx,fhandle      ; Дескриптор файла
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция
                                ; "Дополнить файл" не выполнена
```

Int 21H
Функция 69H(105)
Зарезервирована

Int 21H
Функция 6AH(106)
Зарезервирована

Int 21H
Функция 6BH(107)
Зарезервирована

Int 21H **[4.0]**
Функция 6CH(108)
Расширенное открытие файла

По данному в коде ASCIIZ пути открывает, создает и заменяет файл в указанном или действующем по умолчанию каталоге указанного или действующего по умолчанию диска. Возвращает дескриптор, который можно использовать в программе для последующего доступа к файлу.

При вызове. AH = 6CH
AL = 00H
BX = режим открытия

Бит (ы)	Описание
0–2	тип доступа 000 = только чтение 001 = только запись 010 = чтение-запись
3	зарезервирован (0)
4–6	режим разделения 000 = совместимость 001 = не допускается чтение-запись (не допускается ничего) 010 = не допускается запись

011 = не допускается чтение
100 = разрешено все
7 наследование
0 = дочерний процесс наследует дескриптор
1 = дочерний процесс не наследует дескриптор
8–12 зарезервированы (0)
13 обработка критической ошибки
0 = выполнить прерывание Int 24H
1 = вернуть процессу ошибку.
14 сквозная запись
0 = запись может быть буферизирована и задержана
1 = физическая запись в момент запроса
15 зарезервирован (0)

CX = атрибут файла (биты могут комбинироваться; игнорируется в случае операции открытия)

Бит (ы) Описание (если бит установлен)

0 только для чтения
1 скрытый
2 системный
3 метка тома
4 зарезервировано (0)
5 атрибут архива
6–15 зарезервированы (0)

DX = флаг открытия

Биты Описание

0–3 действие, если файл существует
0000 = не выполняется
0001 = открытие файла
0010 = замена файла
4–7 действие, если файл не существует
0000 = не выполняется
0001 = создание файла
8–15 зарезервированы (0)

DX:SI = сегмент : относительный_адрес пути, заданного в коде ASCIIZ

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AX = дескриптор
CX = полученное действие
1 = файл существовал и был открыт
2 = файл не существовал и был создан
3 = файл существовал и был заменен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Функция не выполняется, если:
– не существует какого-либо элемента пути;
– файл создается в корневом каталоге, который переполнен;
– в заданном каталоге уже существует файл с тем же именем, что имеет и создаваемый файл, и атрибутом *только для чтения*;

– программа выполняется в сети, а пользователь этой программы имеет недостаточные права доступа.

Обычно когда файл создается, он не получает никаких специальных атрибутов (0). Впоследствии атрибут файла может быть изменен с помощью прерывания Int 21H с функцией 43H.

Данная функция объединяет возможности прерывания Int 21H с функциями 3CH, 3DH и 5BH. В систему MS-DOS она была добавлена в целях совместимости с функцией DosOpen системы OS/2.

Пример. Создать файл MYFILE.DAT, если он еще не существует, в каталоге \MYDIR диска C и сохранить дескриптор для последующего доступа к файлу:

```
fname   db      'C:\MYDIR\MYFILE.DAT',0

fhandle dw      ?
        .
        .
        .
        mov     ax,6c00h        ; Номер функции
        mov     bx,4042h        ; Чтение - запись, разрешено все
                                ; Режим сквозного чтения
        xor     cx,cx           ; Без специального атрибута
        mov     dx,0010h        ; Создать, если не существует
                                ; Не выполнять, если существует
        mov     si,seg fname    ; Адрес пути
        mov     ds,si
        mov     si,offset fname
        int     21h             ; Переход в MS-DOS
        jc      error           ; Перейти, если операция открытия
                                ; не выполнена
        mov     fhandle,ax      ; Сохранить дескриптор файла
        .
        .
```

Int 22H [1.0]
Адрес обработчика завершения

Машинный вектор прерывания Int 22H (ячейки памяти от 0000:0088H до 0000:008BH) содержит адрес программы, которая получает управление в тех случаях, когда выполняющаяся в настоящий момент программа завершается через Int 20H, Int 27H и Int 21H с функциями 00H, 31H и 4CH. Кроме того, в момент загрузки программы, но до того, как она начнет выполняться, адрес из этого вектора копируется в префикс программного сегмента (PSP) по относительным адресам от 0AH до 0DH, а восстанавливается адрес из PSP (в случае, если он изменялся прикладной программой) как часть системной процедуры завершения. Это прерывание никогда не следует задавать непосредственно.

Int 23H [1.0]
Адрес обработчика по Ctrl-C

Машинный вектор прерывания Int 23H (ячейки памяти от 0000:008CH до 0000:008FH) содержит адрес программы, которая получает управление в тех случаях, когда обнаруживается Ctrl-C во время выполнения какой-либо функции символьно-ориентированного ввода-вывода или установлен флаг BREAK во время обращения

к большинству других функций MS-DOS. Кроме того, в момент загрузки программы, но до того, как она начнет выполняться, адрес из этого вектора копируется в префикс программного сегмента (PSP) по относительным адресам от 0EH до 11H, а восстанавливается адрес из PSP (в случае, если он изменялся прикладной программой) как часть системной процедуры завершения. Это прерывание никогда не следует задавать непосредственно.

Замечания. Строки инициализации в прикладной программе могут использовать прерывание Int 21H с функцией 25H с целью перенаправить вектор прерывания Int 23H на свою собственную программу обработки Ctrl-C. В этом случае программа может избежать неожиданного завершения в результате ввода пользователем Ctrl-C или Ctrl-Break.

В тех случаях, когда обнаруживается Ctrl-C и программный обработчик прерывания Int 23H получает управление, все регистры сохраняют значения, соответствующие моменту вызова функции ввода-вывода. Обработчик может затем выполнить следующие действия:

- Установить локальный флаг для последующей проверки прикладной программы или совершить какое-нибудь другое действие, после чего выполнить команду IRET. Все регистры должны быть сохранены. Выполняемая функция MS-DOS будет запущена сначала и продолжится до завершения, после чего управление возвращается обычным образом прикладной программе.
- Выполнить соответствующее действие, а затем совершить дальний возврат (RET FAR) с тем, чтобы отдать управление MS-DOS. Система MS-DOS использует состояние флага переноса для определения дальнейших действий. Если флаг переноса установлен, то прикладная программа будет завершена; если флаг переноса сброшен, то прикладная программа будет продолжена обычным образом.
- Сохранить управление, перейдя к программе обработки ошибки внутри прикладной программы, а затем продолжить выполнение программы или выполнить другое запланированное действие, так и не осуществив возврата RET FAR или IRET из процедуры обработки прерывания. Такая последовательность действий не причинит вреда системе.

Внутри тела обработчика прерывания Int 23H можно использовать обращение к любым функциям MS-DOS.

Пример. См. гл. 5.

Int 24H **[1.0]**
Адрес обработчика критической ошибки

Машинный вектор прерывания Int 24H (ячейки памяти 0000:0090H–0000:0093H) содержит адрес программы, которая получает управление в случае обнаружения критической ошибки (обычно аппаратной ошибки). Кроме того, в момент загрузки в программу, но до того, как она начнет выполняться, этот адрес копируется в префикс программного сегмента (PSP) в ячейки 12H–15H, а восстанавливается в адрес из PSP (в случае его изменения прикладной программой) как часть системной процедуры завершения. Никогда не следует задавать это прерывание непосредственно.

Замечания. При входе в обработчик прерывания по критической ошибке бит 7 регистра AH сброшен (0), если произошла ошибка дисковой операции ввода-вывода, в противном случае бит установлен (1). Регистр BP:SI содержит адрес заголовка драйвера устройства, из которого можно получить дополнительную информацию. Прерывания заблокированы. Регистры будут установлены для повторения операции, а код ошибки будет в младшей половине регистра DI, старшая половина которого будет не определена. Младший байт регистра DI содержит:

00H	Попытка записи на защищенный диск
01H	Неизвестное устройство
02H	Дисковод не готов
03H	Неизвестная команда
04H	Ошибка данных (CRC)
05H	Неверная длина структуры запроса
06H	Ошибка установки
07H	Неизвестный тип магнитного носителя
08H	Не найден сектор
09H	Нет бумаги на принтере
0AH	Ошибка записи
0BH	Ошибка чтения
0CH	Общая ошибка
0DH	Зарезервировано
0EH	Зарезервировано
0FH	Неправильная смена диска (MS-DOS только версии 3)

Заметим, что это те же самые коды ошибки, которые возвращаются драйвером в заголовок запроса. Кроме того, после записи стек установлен так, как показано на рис. 8.8.

В тех случаях, когда встречается ошибка дисковой операции ввода-вывода, система MS-DOS автоматически повторяет операцию, прежде чем выдать прерывание по критической ошибке Int 24H. Число повторений варьируется в различных версиях MS-DOS, но обычно заключено в пределах три–пять. Обработчики прерывания Int 24H должны сохранять содержимое регистров SS, SP, DS, ES, BX, CX и DX. Обработчик прерывания Int 24H может использовать только прерывание Int 21H с функциями 01H–0CH и 59H; обращение к другим функциям будет разрушать стек MS-DOS и лишит систему способности выбирать между повторением операции или игнорированием ошибки.

При выдаче команды IRET обработчик прерывания Int 24H должен вернуть в регистр AL код действия, интерпретируемый DOS следующим образом:

0	Игнорировать ошибку
1	Повторить операцию
2	Завершить программу
3	[3.0+] Завершить выполняемую функцию

Если обработчик прерывания возвращает управление прямо в прикладную программу, а не в MS-DOS, он должен восстановить программные регистры, удалив из стека все, за исключением трех последних слов, и выдать команду IRET. Управление передается команде, следующей непосредственно за обращением к функции, которая вызвала ошибку. Эти действия оставляют MS-DOS в нестабильном состоянии до тех пор, пока не будет выполнен вызов прерывания Int 21H с функцией старше, чем 0CH.
Пример. См. гл. 8.

Int 25H **[1.0]**
Абсолютное чтение диска

Обеспечивает непосредственную связь с модулем BIOS MS-DOS в целях чтения данных из логического сектора диска в память.
При вызове. Для доступа к разделам $\leqslant$ 32 Мбайт

AL = номер дисковода (0 = A, 1 = B и т. д.)
CX = число секторов для чтения
DX = начальный номер сектора
DS:BX = сегмент : относительный_адрес буфера
Для доступа к разделам > 32 Мбайт (MS-DOS версий 4.0 и старше)
AL = номер дисковода (0 = A, 1 = B и т. д.)
CX = −1
DS:BX = сегмент : относительный_адрес блока параметров
(см. *Замечания*)

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки (см. *Замечания*)

Замечания. Содержимое всех регистров, за исключением сегментных, может быть изменено.

При возврате из данной функции флаги центрального процессора, первоначально помещенные с помощью прерывания Int 25H на стек, по-прежнему там же и находятся. В целях предотвращения неуправляемого увеличения стека и обеспечения доступа ко всем другим величинам, которые были помещены на стек до вызова к прерыванию Int 25H, необходимо с помощью команды POPF или ADD SP,2 очистить стек.

Логические номера секторов образуются путем последовательной нумерации каждого сектора диска, начиная с цилиндра 0, головки 0, сектора 1 и до самого последнего сектора на диске. Сначала выполняется инкремент номера головки, затем – номера дорожки. Секторы, соседние логически, могут не быть соседними физически ввиду чередования, вводимого на уровне адаптера устройства при работе с дисками некоторых типов.

Код ошибки интерпретируется следующим образом: младший байт (AL) совпадает с кодом ошибки, возвращаемым в младший байт регистра DI, в тех случаях, когда запрашивается прерывание Int 24H. Старший байт (AH) содержит:

01H, если неправильная команда
02H, если неправильная метка адреса
04H, если запрашиваемый сектор не найден
08H, если не выполнен прямой доступ к памяти (direct memory access, DMA).
10H, если ошибка данных (ошибка CRC)
20H, если не сработал контроллер
40H, если не выполнена операция установки
80H, если подсоединенное устройство не среагировало

[4.0+] В среде MS-DOS версии 4 в тех случаях, когда разделы, к которым происходит обращение, больше 32 Мбайт, данная функция использует блок параметров следующего формата:

Байты	Описание
00H–03H	32-битный номер сектора
04H–05H	Число секторов, предназначенных для чтения
06H–07H	Относительный адрес буфера
08H–09H	Сегмент буфера

Пример. Прочитать в область памяти buff логический сектор 1 диска A. (Дискеты

большинства версий системы MS-DOS содержат в этом секторе начало таблицы размещения файлов.)

```
buff    db      512 dup (?)     ; Получить данные с диска
        .
        .
        .
        mov     al,0            ; Дисковод A
        mov     cx,1            ; Номер сектора
        mov     dx,1            ; Номер начального сектора
        mov     bx,seg buff     ; Адрес буфера
        mov     ds,bx
        mov     bx,offset buff
        int     25h             ; Запрос на операцию чтения диска
        jc      error           ; Перейти, если операция чтения не выполнена
        add     sp,2            ; Очистить стек
        .
        .
        .
```

Int 26H **[1.0]**
Абсолютная запись на диск

Обеспечивает непосредственную связь с модулем BIOS MS-DOS в целях записи в логический сектор диска данных из памяти.

При вызове. Для доступа к разделам ⩽ 32 Мбайт
AL = номер дисковода (0 = A, 1 = B и т. д.)
CX = число секторов для записи
DX = начальный номер сектора
DS:BX = сегмент : относительный_адрес буфера
Для доступа к разделам > 32 Мбайт (MS-DOS версий 4.0 и старше)
AL = номер дисковода (0 = A, 1 = B и т. д.)
CX = − 1
DS:BX = сегмент : относительный_адрес блока параметров
(см. *Замечания*)

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки (см. *Замечания*)

Замечания. Содержимое всех регистров, за исключением сегментных, может быть разрушено.

При возврате из данной функции флаги центрального процессора, первоначально помещенные с помощью прерывания Int 26H на стек, по-прежнему там же и находятся. В целях предотвращения неуправляемого увеличения стека и обеспечения доступности ко всем другим величинам, которые были помещены на стек до вызова прерывания Int 26H, необходимо с помощью команды POPF или ADD SP,2 очистить стек.

Логические номера секторов образуются путем последовательной нумерации каждого сектора диска, начиная с цилиндра 0, головки 0, сектора 1 и до самого последнего сектора на диске. Сначала выполняется инкремент номера головки, затем – номера дорожки. Секторы, соседние логически, могут не быть соседними физически ввиду чередования, вводимого на уровне адаптера устройства при работе с дисками некоторых типов.

Код ошибки интерпретируется следующим образом: младший байт (AL) совпадает с кодом ошибки, возвращаемым в младший байт регистра DI, в тех случаях, когда запрашивается прерывание Int 24H.
Старший байт (AH) содержит:

01H, если неправильная команда
02H, если неправильная метка адреса
03H, если произведена попытка записи на защищенный диск
04H, если запрашиваемый сектор не найден
08H, если не выполнен прямой доступ к памяти (DMA)
10H, если ошибка данных (ошибка CRC)
20H, если не сработал контроллер
40H, если не выполнена операция установки
80H, если подсоединенное устройство не среагировало

[4.0+] В среде MS-DOS версии 4 в тех случаях, когда разделы, к которым происходит обращение, больше 32 Мбайт, данная функция использует блок параметров следующего формата:

Байты	Описание
00H–03H	32-битный номер сектора
04H–05H	Число секторов, предназначенных для чтения
06H–07H	Относительный адрес буфера
08H–09H	Сегмент буфера

Пример. Записать содержимое области памяти с именем buff в логический сектор 3 диска C.
Предупреждение. Запуск приведенной ниже программы может повредить файловую систему на вашем жестком диске. К сожалению, невозможно предложить вполне безопасный пример этой функции.

```
buff    db      512 dup (?)     ; Данные для записи
        .
        .
        .
        mov     al,2            ; Дисковод C
        mov     cx,1            ; Номер сектора
        mov     dx,3            ; Номер начального сектора
        mov     bx,seg buff     ; Адрес буфера
        mov     ds,bx
        mov     bx,offset buff
        int     26h             ; Запрос на операцию вывода записи на диск
        jc      error           ; Перейти, если операция вывода записи
                                ; не выполнена
        add     sp,2            ; Очистить стек
        .
        .
        .
```

Int 27H [1.0]
Завершить и оставить резидентной

Завершает выполнение текущей исполняемой программы и резервирует часть или всю память с целью не допустить ее перекрытия следующей загружаемой транзитной программой. Затем система MS-DOS выполняет следующие действия:

● Файловые буферы сбрасываются, а все дескрипторы файлов или устройств, открытые самим процессом, закрываются.

● Вектор обработчика завершения (прерывание Int 22H) восстанавливается по содержимому PSP:000AH.

● Вектор обработчика по Ctrl-C (прерывание Int 23H) восстанавливается по содержимому PSP:000EH.

● [2.0+] Вектор обработчика по критической ошибке (прерывание Int 24H) восстанавливается по содержимому PSP:0012H.

● Управление передается обработчику завершения.

В тех случаях, когда программа возвращается в программу COMMAND.COM, управление передается в резидентную часть последней, а ее транзитная часть, если это необходимо, перезагружается. В тех случаях, когда выполняется командный файл, выбирается и интерпретируется следующая строка файла; в противном случае выдается запрос на следующую команду пользователя.

При вызове. DX = относительный адрес последнего байта защищаемой программы плюс один (относительно префикса программного сегмента)

CX = сегмент префикса программного сегмента

При возврате. Ничего не возвращается.

Замечания. Обычно эта функция используется с целью обеспечить возможность написанным пользователем утилитам, драйверам и обработчикам прерываний загружаться как обычным программам типа .COM или .EXE и оставаться после этого резидентными. Последующий вход в программу осуществляется через аппаратное или системное прерывание.

Данная функция пытается установить начальный блок выделенной памяти длиной (в байтах), задаваемой содержимым регистра DX. Если в прикладной программе через прерывание Int 21H были уже затребованы другие блоки памяти, то они с помощью данной функции не будут освобождены.

Другие способы осуществления окончательного выхода:

– с помощью прерывания Int 20H;
– с помощью прерывания Int 21H с функцией 00H;
– с помощью прерывания Int 21H с функцией 31H;
– с помощью прерывания Int 21H с функцией 4CH.

Не следует вызывать данную функцию из программ типа .EXE, загружаемых в верхней части области транзитных программ (т.е. компонуемых с помощью ключа /HIGH), так как при этом резервируется память, которая обычно используется транзитной частью COMMAND.COM. Если программу COMMAND.COM нельзя перезагрузить, то система выйдет из строя.

Если регистр DX содержит значения в диапазоне 0FFF1H–0FFFFH, данная функция работает некорректно. В этом случае система MS-DOS не учитывает старший бит находящегося в регистре DX значения, что приводит к резервированию памяти на 32 Кбайт меньше, чем это было запрошено программой.

[2.0+] Предпочтительнее вместо этой функции использовать прерывание Int 21H с функцией 31H, которое поддерживает коды возврата, что позволяет резервировать больший объем памяти и не требует, чтобы в регистре CS находился адрес сегмента префикса программного сегмента.

[3.0+] Если программа выполняется в сети, то до ее завершения необходимо освободить все ранее запертые районы файла.

Пример. Завершить и оставить резидентной, зарезервировав при этом достаточно памяти для хранения всей программы:

```
        .
        .
        .
        mov     dx,offset pend  ; DX = сколько  байтов сохранить
        int     27h             ; Завершить, оставить резидентной
        .
        .
        .
pend    equ     $               ; Относительный адрес, конец программы

        end
```

Int 28H
Зарезервировано

Int 29H
Зарезервировано

Int 2AH
Зарезервировано

Int 2BH
Зарезервировано

Int 2CH
Зарезервировано

Int 2DH
Зарезервировано

Int 2EH
Зарезервировано

Int 2FH **[3.0]**
Мультиплексное прерывание

Обеспечивает взаимодействие с другими процессами и расширениями MS-DOS, такими, как спулер печати, программами ASSIGN, SHARE и APPEND. Мультиплексный номер в регистре AH задает процесс или расширение, с которыми организуется взаимодействие. Диапазон значений 00H–BFH резервируется для системы MS-DOS; прикладные программы могут использовать диапазон номеров C0H–FFH.

Int 2FH **[3.0]**
Функция 01H
Спулер печати

Помещает файл в спулер печати, удаляет файл из очереди к спулеру или возвращает состояние принтера. Спулер печати, который находится в файле PRINT.COM, впервые был добавлен в систему MS-DOS версии 2.0, однако интерфейс прикладной программы к спулеру не был документирован до версии 3 системы MS-DOS.
При вызове. AH = 01H
AL = подфункция
00H = получить установленное состояние

01H = послать файл на печать
02H = удалить файл из очереди на печать
03H = удалить из очереди все файлы
04H = приостановить выполнение заданий на печать до чтения состояния
05H = продолжить выполнение
DS:DX = сегмент : относительный_адрес пакета (подфункция 01H)
сегмент : относительный_адрес пути, заданного в коде ASCIIZ (подфункция 02H)

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
и, если при вызове AL = 00H
AL = состояние спулера печати
00H, если не установлен, добро на установку
01H, если не установлен, установка не разрешается
FFh, если установлен
или, если при вызове AL = 04H
DX = счетчик ошибки
DS:SI = сегмент : относительный_адрес таблицы файлов, стоящих в очереди на печать
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечания. Текст, передаваемый подфункции 01H, состоит из пяти байтов. Первый байт содержит уровень, который должен быть 00H для текущей версии MS-DOS. Последующие четыре байта содержат сегмент : относительный_адрес пути, заданного в коде ASCIIZ, в который не могут включаться шаблоны групповых операций. Если заданный файл существует, то он добавляется в очередь на печать.

Шаблоны групповых операций «*» и «?» могут включаться в описание пути, передаваемого подфункции 02H, что делает возможным за одно обращение к функции удалять из очереди на печать группу файлов.

Адрес, возвращаемый подфункцией 04H, указывает на таблицу, состоящую из 64-байт записей, каждая из которых содержит путь, заданный в коде ASCIIZ. Первым в таблице располагается путь к файлу, печатаемому в настоящий момент. Последней записью в таблице является нулевая строка (один байт 00H).

Int 2FH [3.2]
Функция 02H
ASSIGN

Возвращает код, который указывает, была ли загружена резидентная часть утилиты ASSIGN.

При вызове. AH = 02H
AL = подфункция
00H = получить состояние установки

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AL = состояние установки утилиты ASSIGN

00H, если не установлена,
добро на установку
01H, если не установлена
установка не разрешается
FFH, если установлена
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Int 2FH [3.2]
Функция 10H(16)
SHARE

Возвращает код, который указывает, был ли загружен модуль разделения файлов SHARE.EXE.

При вызове. AH = 10H
AL = подфункция
00H = получить состояние установки

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
AL = состояние установки модуля SHARE
00H, если не установлен, добро на установку
01H, если не установлен, установка не разрешается
FFH, если установлен
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Int 2FH [3.3]
Функция B7H(183)
APPEND

Дает возможность прикладной программе проверить, была ли произведена установка модуля APPEND. Если модуль APPEND резидентен, то возвращается номер версии APPEND, его состояние и путь, используемый для поиска файлов с данными.

При вызове. AH = B7H
AL = подфункция
00H = получить состояние установки
02H = получить номер версии APPEND (4.0)
04H = получить указатель пути APPEND (4.0)
06H = получить состояние функции APPEND (4.0)
07H = установить состояние функции APPEND (4.0)
11H = установить состояние возвращения найденного имени (4.0, см. *Замечание*)
BX = состояние APPEND (если AL = 07H)

Бит (ы)	**Описание** (если бит установлен)
0	средство APPEND разблокировано
1–12	зарезервированы (0)
13	действует ключ /PATH

14 действует ключ /E
15 действует ключ /X

При возврате. Если функция успешно выполнена
Флаг переноса = сброшен
и, если при вызове AL = 00H
AL = статус установки APPEND
000H, если не установлен, добро на установку
01H, если не установлен, установка не разрешена
FFH, если установлен
или, при вызове AL = 02H (MS-DOS 4.0)
AX = FFFFH, если APPEND системы MS-DOS 4.0
или, если при вызове AL = 04H (MS-DOS 4.0)
ES:DI = сегмент : относительный_адрес активного пути APPEND
или, если при вызове AL = 06H (MS-DOS 4.0)
BX = состояние APPEND (см. выше)
Если функция не выполнена
Флаг переноса = установлен
AX = код ошибки

Замечание. Если состояние возвращения найденного имени устанавливается с помощью подфункции 11H, то в следующую прикладную программу при обращении к прерыванию Int 21H с функциями 3DH, 43H или 6CH возвращается полностью квалифицированное имя файла. Имя помещается по тому же адресу, что и строка параметров кода ASCIIZ для функции прерывания Int 21H, так что прикладная программа должна предоставить буфер адекватного размера. Статус возвращения найденного имени сбрасывается после обработки расширением APPEND следующего функционального вызова Int 21H.

Раздел 3. Справочник по функциям ПЗУ BIOS фирмы IBM и манипулятора мышь

Пояснения для читателя

В заголовках функциональных вызовов видеодрайвера ПЗУ BIOS (Int 10H) используются следующие обозначения:

[MDA]	Адаптер монохромного дисплея
[CGA]	Цветной графический адаптер
[PCjr]	Видеоконтроллер системной платы машины PCjr
[EGA]	Улучшенный графический адаптер
[MCGA]	Многоцветная графическая матрица (модели PS/2 25 и 35)
[VGA]	Видеографическая матрица (модели PS/2 50 и выше)

В настоящем разделе используются также следующие обозначения:

[PC]	Исходные компьютеры IBM PC, PC/XT и PCjr, если не указано иное
[AT]	PC/AT и PC/XT-286, если не указано иное
[PS/2]	Все модели PS/2 (включая модели 25 и 30), если не указано иное

Функции ПЗУ BIOS, используемые только в машинах PC Convertible, опущены.

Некоторые функции на конкретных машинах поддерживаются микросхемами ПЗУ BIOS только самых последних выпусков (например, Int 1AH с функциями 00H и 01H на PC/XT). Как правило, описания функций не содержат условных обозначений таких машин, поскольку программа не может по содержимому байта (или байтов) идентификации машины достоверно определить наличие этих функций.

Список функциональных вызовов ПЗУ BIOS и манипулятора мышь

Int	Функция	Подфункция	Описание
10H			Видеодрайвер
10H	00H		Установить видеорежим
10H	01H		Установить тип курсора
10H	02H		Установить позицию курсора
10H	03H		Получить позицию курсора
10H	04H		Получить позицию светового пера
10H	05H		Установить видеостраницу
10H	06H		Инициализировать или прокрутить окно вверх
10H	07H		Инициализировать или прокрутить окно вниз
10H	08H		Прочитать символ и атрибут в позиции курсора
10H	09H		Записать символ и атрибут в позицию курсора
10H	0AH(10)		Записать символ в позицию курсора
10H	0BH(11)		Установить палитру, фон или край экрана
10H	0CH(12)		Записать графический пиксел
10H	0DH(13)		Прочитать графический пиксел
10H	0EH(14)		Записать символ в режиме телетайпа
10H	0FH(15)		Получить видеорежим
10H	10H(16)	00H	Установить регистр палитры
10H	10H(16)	01H	Установить цвет края экрана

Продолжение таблицы

Int	Функция	Подфункция	Описание
10H	10H(16)	02H	Установить палитру и край экрана
10H	10H(16)	03H	Переключить бит мерцание/яркость
10H	10H(16)	07H	Получить регистр палитры
10H	10H(16)	08H	Получить цвет края экрана
10H	10H(16)	09H	Получить палитру и край экрана
10H	10H(16)	10H(16)	Установить регистр цвета
10H	10H(16)	12H(18)	Установить блок регистров цветов
10H	10H(16)	13H(19)	Установить состояние цветовой страницы
10H	10H(16)	15H(21)	Получить регистр цвета
10H	10H(16)	17H(23)	Получить блок регистров цветов
10H	10H(16)	1AH(26)	Получить состояние цветовой страницы
10H	10H(16)	1BH(27)	Установить значение яркости полутонового изображения
10H	11H(17)	00H	Загрузить шрифт пользователя
10H	11H(17)	01H	Загрузить шрифт ПЗУ 8 × 14
10H	11H(17)	02H	Загрузить шрифт ПЗУ 8 × 8
10H	11H(17)	03H	Установить описатель блоков
10H	11H(17)	04H	Загрузить шрифт ПЗУ 8 × 16
10H	11H(17)	10H(16)	Загрузить шрифт пользователя, перепрограммировать контроллер
10H	11H(17)	11H(17)	Загрузить шрифт ПЗУ 8 × 14, перепрограммировать контроллер
10H	11H(17)	12H(18)	Загрузить шрифт ПЗУ 8 × 8, перепрограммировать контроллер
10H	11H(17)	14H(20)	Загрузить шрифт ПЗУ 8 × 16, перепрограммировать контроллер
10H	11H(17)	20H(32)	Установить указатель шрифта Int 1FH
10H	11H(17)	21H(33)	Настроить Int 43H на шрифт пользователя
10H	11H(17)	22H(34)	Настроить Int 43H на шрифт ПЗУ 8 × 14
10H	11H(17)	23H(35)	Настроить Int 43H на шрифт ПЗУ 8 × 8
10H	11H(17)	24H(36)	Настроить Int 43H на шрифт ПЗУ 8 × 16
10H	11H(17)	30H(48)	Получить информацию о шрифте
10H	12H(18)	10H(16)	Получить информацию о конфигурации
10H	12H(18)	20H(32)	Выбрать дополнительную программу печати экрана
10H	12H(18)	30H(48)	Установить число строк развертки
10H	12H(18)	31H(49)	Разрешить/запретить загрузку палитры
10H	12H(18)	32H(50)	Разрешить/запретить функционирование видеосистемы
10H	12H(18)	33H(51)	Разрешить/запретить определение яркости полутонового изображения
10H	12H(18)	34H(52)	Разрешить/запретить эмуляцию курсора
10H	12H(18)	35H(53)	Переключить активную видеосистему
10H	12H(18)	36H(54)	Разрешить/запретить регенерацию экрана
10H	13H(19)		Записать строку в режиме телетайпа
10H	1AH(26)		Получить или установить код конфигурации видеосистемы
10H	1BH(27)		Получить информацию о функциональной поддержке/состоянии видеосистемы
10H	1CH(28)		Сохранить или восстановить состояние видеосистемы
11H			Получить конфигурацию оборудования
12H			Получить размер стандартной памяти
13H			Драйвер диска
13H	00H		Привести в исходное состояние дисковую систему
13H	01H		Получить состояние дисковой системы
13H	02H		Прочитать сектор
13H	03H		Записать сектор
13H	04H		Верифицировать сектор
13H	05H		Форматировать дорожку

Продолжение таблицы

Int	Функция	Подфункция	Описание
13H	06H		Форматировать дефектную дорожку
13H	07H		Форматировать дисковод
13H	08H		Получить параметры дисковода
13H	09H		Инициализировать характеристики жесткого диска
13H	0AH(10)		Прочитать расширенный сектор
13H	0BH(11)		Записать расширенный сектор
13H	0CH(12)		Установка головок
13H	0DH(13)		Привести в исходное состояние систему жесткого диска
13H	0EH(14)		Прочитать буфер сектора
13H	0FH(15)		Записать буфер сектора
13H	10H(16)		Получить состояние дисковода
13H	11H(17)		Привести в исходное состояние позиционер дисковода
13H	12H(18)		Диагностика ОЗУ контроллера
13H	13H(19)		Диагностика контроллера дисковода
13H	14H(20)		Внутренняя диагностика контроллера
13H	15H(21)		Получить тип диска
13H	16H(22)		Получить состояние смены диска
13H	17H(23)		Установить тип диска
13H	18H(24)		Установить тип носителя для форматирования
13H	19H(25)		Парковать головки
13H	1AH(26)		Форматировать дисковод ESDI
14H			Драйвер последовательного коммуникационного порта
14H	00H		Инициализировать коммуникационный порт
14H	01H		Записать символ в коммуникационный порт
14H	02H		Прочитать символ из коммуникационного порта
14H	03H		Получить состояние коммуникационного порта
14H	04H		Расширенная инициализация коммуникационного порта
14H	05H		Расширенное управление коммуникационным портом
15H			Расширения подсистемы ввода-вывода
15H	00H		Включить мотор кассеты
15H	01H		Выключить мотор кассеты
15H	02H		Прочитать кассету
15H	03H		Записать кассету
15H	0FH(15)		Периодические прерывания при форматировании ESDI-дисковода
15H	21H(33)	00H	Прочитать журнал ошибок POST
15H	21H(33)	01H	Записать журнал ошибок POST
15H	4FH(79)		Перехват клавиатуры
15H	80H(128)		Устройство открыто
15H	81H(129)		Устройство закрыто
15H	82H(130)		Завершение процесса
15H	83H(131)		Ожидание события
15H	84H(132)		Прочитать джойстик
15H	85H(133)		Клавиша SysReq
15H	86H(134)		Задержка
15H	87H(135)		Переслать блок расширенной памяти
15H	88H(136)		Получить размер расширенной памяти
15H	89H(137)		Вход в защищенный режим
15H	90H(144)		Ожидание устройства
15H	91H(145)		Устройство закончило
15H	C0H(192)		Получить системную среду
15H	C1H(193)		Получить адрес расширенной области данных BIOS
15H	C2H(194)	00H	Разблокировать/заблокировать координатное устройство

Продолжение таблицы

Int	Функция	Подфункция	Описание
15H	C2H(194)	01H	Привести в исходное состояние координатное устройство
15H	C2H(194)	02H	Установить частоту выборки
15H	C2H(194)	03H	Установить разрешение
15H	C2H(194)	04H	Получить тип координатного устройства
15H	C2H(194)	05H	Инициализировать интерфейс координатного устройства
15H	C2H(194)	06H	Установить масштаб или получить состояние
15H	C2H(194)	07H	Установить адрес обработчика прерываний от координатного устройства
15H	C3H(195)		Установить сторожевой тайм-аут
15H	C4H(196)		Выбор программируемых опций
16H			Драйвер клавиатуры
16H	00H		Прочитать символ с клавиатуры
16H	01H		Получить состояние клавиатуры
16H	02H		Получить флаги клавиатуры
16H	03H		Установить скорость повторения
16H	04H		Установить звуковой сигнал клавиш
16H	05H		Протолкнуть символ и скен-код
16H	10H(16)		Прочитать символ с расширенной клавиатуры
16H	11H(17)		Получить состояние расширенной клавиатуры
16H	12H(18)		Получить флаги расширенной клавиатуры
17H			Драйвер принтера параллельного порта
17H	00H		Записать символ на принтер
17H	01H		Инициализировать порт принтера
17H	02H		Получить состояние принтера
18H			ПЗУ системы Бейсик
19H			Перезагрузка системы
1AH			Драйвер часов реального времени (КМОП)
1AH	00H		Получить число тактов
1AH	01H		Установить число тактов
1AH	02H		Получить время
1AH	03H		Установить время
1AH	04H		Получить дату
1AH	05H		Установить дату
1AH	06H		Установить будильник
1AH	07H		Сбросить будильник
1AH	0AH(10)		Получить содержимое счетчика дней
1AH	0BH(11)		Установить счетчик дней
1AH	80H(128)		Установить источник звука
33H			Драйвер мыши фирмы Microsoft
33H	00H		Привести мышь в исходное состояние и получить состояние драйвера
33H	01H		Вывести на экран указатель мыши
33H	02H		Погасить указатель мыши
33H	03H		Получить положение мыши и состояние клавиш
33H	04H		Установить положение указателя мыши
33H	05H		Получить информацию о нажатиях клавиш мыши
33H	06H		Получить информацию об отпусканиях клавиш мыши
33H	07H		Установить горизонтальные границы для указателя
33H	08H		Установить вертикальные границы для указателя
33H	09H		Установить форму графического указателя
33H	0AH(10)		Установить тип текстового указателя
33H	0BH(11)		Прочитать состояние счетчиков движения мыши
33H	0CH(12)		Установить определяемый пользователем обработчик событий мыши
33H	0DH(13)		Включить эмуляцию светового пера

Продолжение таблицы

Int	Функция	Подфункция	Описание
33H	0EH(14)		Выключить эмуляцию светового пера
33H	0FH(15)		Установить отношение микки на пиксел
33H	10H(16)		Установить область подавления указателя мыши
33H	13H(19)		Установить порог удвоения скорости
33H	14H(20)		Сменить определяемые пользователем обработчики событий мыши
33H	15H(21)		Получить размер буфера хранения состояния мыши
33H	16H(22)		Сохранить состояние драйвера мыши
33H	17H(23)		Восстановить состояние драйвера мыши
33H	18H(24)		Установить дополнительный обработчик событий мыши
33H	19H(25)		Получить адрес дополнительного обработчика событий мыши
33H	1AH(26)		Установить чувствительность мыши
33H	1BH(27)		Получить чувствительность мыши
33H	1CH(28)		Установить частоту прерываний от мыши
33H	1DH(29)		Выбрать страницу указателя
33H	1EH(30)		Получить страницу указателя
33H	1FH(31)		Заблокировать драйвер мыши
33H	20H(32)		Разблокировать драйвер мыши
33H	21H(33)		Привести в исходное состояние драйвер мыши
33H	22H(34)		Установить язык сообщений драйвера мыши
33H	23H(35)		Получить номер языка
33H	24H(36)		Получить информацию о мыши

Int 10H
Функция 00H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Установить видеорежим

Выбирает текущий режим видеосистемы. Кроме того, при наличии более одного видеоконтроллера выбирает активный видеоконтроллер.

При вызове. AH = 00H
AL = видеорежим (см. *Замечания*)

При возврате. Ничего не возвращается.

Замечания. В различных моделях машин IBM и видеоадаптеров используются следующие видеорежимы:

Режим	Разрешение	Цвета	Текст графика	MDA	CGA	PCjr	EGA	MCGA	VGA
00H	40 × 25	16	Текст		*	*	*	*	*
	Смешивание цветов выключено								
01H	40 × 25	16	Текст		*	*	*	*	*
02H	80 × 25	16	Текст		*	*	*	*	*
	Смешивание цветов выключено								
03H	80 × 25	16	Текст		*	*	*	*	*
04H	320 × 200	4	Графика		*	*	*	*	*
05H	320 × 200	4	Графика		*	*	*	*	*
	Смешивание цветов выключено								
06H	640 × 200	2	Графика		*	*	*	*	*
07H	80 × 25	2[1)]	Текст	*			*		*
08H	160 × 200	16	Графика			*			
09H	320 × 200	16	Графика			*			
0AH	640 × 200	4	Графика			*			
0BH	Зарезервировано								

Продолжение таблицы

Режим	Разрешение	Цвета	Текст графика	MDA	CGA	PCjr	EGA	MCGA	VGA
0CH	Зарезервировано								
0DH	320 × 200	16	Графика				*		*
0EH	640 × 200	16	Графика				*		*
0FH	640 × 350	2[2)]	Графика				*		*
10H	640 × 350	4	Графика				*[3)]		
10H	640 × 350	16	Графика				*[4)]		*
11H	640 × 480	2	Графика					*	*
12H	640 × 480	16	Графика						*
13H	320 × 200	256	Графика					*	*

[1)] Только для монохромного монитора.
[2)] Только для монохромного монитора.
[3)] EGA с ОЗУ объемом 64 Кбайт.
[4)] EGA с ОЗУ объемом 128 Кбайт или более.

Наличие или отсутствие смешивания цветов для получения сложного аналогового цветового сигнала имеет значение только при использовании композитного монитора. Для мониторов RGB нет функционального отличия режимов 00H и 01H или 02H и 03H. С адаптером CGA в режиме 04H доступны две палитры, а в режиме 05H – одна.

На машинах PC/AT, PCjr и PS/2 при выборе нового режима видеобуфер не очищается, если установлен бит 7 AL. На машинах PC или PC/XT эта возможность предоставляется, только если установлены видеоплаты EGA или VGA (имеющие собственные ПЗУ BIOS).

Int 10H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 01H
Установить тип курсора

Выбирает начальную и конечную строки развертки для мерцающего аппаратного курсора в текстовых режимах.

При вызове. AH = 01H
CH, биты 0–4 = начальная строка курсора
CL, биты 0–4 = конечная строка курсора

При возврате. Ничего не возвращается.

Замечания. В текстовых режимах аппаратные видеосредства заставляют курсор мерцать, и убрать мерцание нельзя. В графических режимах аппаратный курсор недоступен.

Значения по умолчанию, устанавливаемые из ПЗУ BIOS:

Режим	**Начало**	**Конец**
Монохромный режим 07H	11	12
Текстовые режимы 00H–03H	6	7

При использовании видеоплат EGA, MCGA и VGA в текстовых режимах 00H–03H ПЗУ BIOS воспринимает значения начала и конца курсора, как если бы матрица символа имела размер 8 × 8, а затем отображает эти значения в соответствии с реальными размерами матрицы символа. Это отображение носит название эмуляция курсора.

Имеются разные способы гашения курсора. В адаптерах MDA, CGA и VGA

установка регистра CH = 20H вызывает исчезновение курсора. Метод установки не имеющих смысла для текущего видеорежима значений начала и конца курсора ненадежен. Можно позиционировать курсор по неизображаемому адресу, например (x, y) = (0,25).

Int 10H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 02H
Установить позицию курсора

Позиционирует курсор на экране с использованием текстовых координат.
При вызове. AH = 02H
BH = страница
DH = строка (y-координата)
DL = столбец (x-координата)
При возврате. Ничего не возвращается.
Замечания. Для каждой видеостраницы организуется отдельный курсор, и с помощью настоящей функции можно установить любой из этих курсоров независимо от того, какая страница активна. Число доступных видеостраниц зависит от видеоадаптера и текущего видеорежима. (См. Int 10H с функцией 05H.)

Текстовые координаты (x, y) = (0, 0) соответствуют верхнему левому углу экрана.

Максимальное значение каждой текстовой координаты зависит от видеоадаптера и текущего видеорежима следующим образом:

Режим	Максимальное значение x	Максимальное значение y
00H	39	24
01H	39	24
02H	79	24
03H	79	24
04H	39	24
05H	39	24
06H	79	24
07H	79	24
08H	19	24
09H	39	24
0AH	79	24
0BH	Зарезервировано	
0CH	Зарезервировано	
0DH	39	24
OEH	79	24
OFH	79	24
10H	79	24
11H	79	29
12H	79	29
13H	39	24

Int 10H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 03H
Получить позицию курсора

Возвращает текущую позицию курсора на экране в текстовых координатах.

При вызове. AH = 03H
BH = страница
При возврате. CH = начальная строка развертки курсора
CL = конечная строка развертки курсора
DH = строка (y-координата)
DL = столбец (x-координата)
Замечание. Для каждой видеостраницы организуется отдельный курсор, и с помощью настоящей функции можно исследовать любой из этих курсоров независимо от того, какая страница активна. Число доступных видеостраниц зависит от видеоадаптера и текущего видеорежима. (См. Int 10H с функцией 05H.)

Int 10H **[CGA] [PCjr] [EGA]**
Функция 04H
Получить позицию светового пера

Возвращает текущее состояние и позицию светового пера.
При вызове. AH = 04H
При возврате. AH = 00H, если перо не опущено/не включено
01H, если перо опущено/включено
BX = пиксел-столбец (графическая x-координата)
CH = пиксел-строка (графическая y-координата, режимы 04H–06H)
CX = пиксел-строка (графическая y-координата, режимы 0DH–13H)
DH = символьная строка (текстовая y-координата)
DL = символьный столбец (текстовая x-координата)
Замечания. Диапазон текстовых и графических координат, возвращаемых этой функцией, зависит от текущего видеорежима.

При использовании адаптера CGA возвращаемые функцией графические координаты не являются непрерывными. Y-координата всегда кратна двум; x-координата либо кратна четырем (для видеорежимов 320 × 200), либо кратна восьми (для видеорежимов 640 × 200).

Для получения максимальной чувствительности светового пера по всей ширине экрана следует аккуратно выбирать цвета изображения и фона.

Int 10H **[CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 05H
Установить видеостраницу

Выбирает активную видеостраницу видеосистемы.
При вызове. Для CGA, EGA, MCGA, VGA
AH = 05H
AL = страница
0–7 для режимов 00H и 01H
0–3 для режимов 02H и 03H
0–7 для режимов 02H и 03H
0–7 для режима 07H
0–7 для режима 0DH
0–3 для режима 0EH
0–1 для режима 0FH
0–1 для режима 10H
Только для PCjr
AH = 05H

AL = подфункция
80H = прочитать регистры страниц дисплея/ЦП
81H = установить регистр страницы ЦП
82H = установить регистр страницы дисплея
83H = установить регистры страниц ЦП и дисплея
BH = страница дисплея (подфункции 82H и 83H)
BL = страница ЦП (подфункции 81H и 83H)

При возврате. С адаптерами CGA, MCGA, VGA
Ничего не возвращается
С PCjr, если при вызове функции AL = 80H–83H
BH = регистр страницы дисплея
BL = регистр страницы ЦП

Замечания. Не указанные выше комбинации видеорежимов и адаптеров поддерживают одну видеостраницу (например, монохромный адаптер в режиме 7).

Переключение страниц не изменяет их содержимого. На любую видеостраницу независимо от того, какая страница отображается в настоящий момент, может быть записан текст с помощью прерывания Int 10H с функциями 02H, 09H и 0AH.

При работе с PCjr страница ЦП определяет часть области памяти 00000H–1FFFFH, которая будет аппаратно отображаться на 16 Кбайт памяти, начинающихся с сегмента B800H. Страница дисплея определяет начальный адрес физической памяти, используемой видеоконтроллером для регенерации изображения. С помощью этих регистров можно добиться эффектов «оживления». Программам, непосредственно записывающим в сегмент B800H, доступны только первые 16 Кбайт буфера видеорегенерации. Программы, требующие прямого доступа ко всему пространству 32-Кбайт буфера в режимах 09H и 0AH, могут получить текущую страницу ЦП из переменной PAGDAT ПЗУ BIOS по адресу 0040:008AH.

Int 10H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 06H
Инициализировать или прокрутить окно вверх

Инициализирует указанное видеоокно, заполняя его символами пробелов ASCII с заданным атрибутом, либо прокручивает содержимое окна вверх на заданное число строк.

При вызове. AH = 06H
AL = число строк прокрутки (если 0, все окно очищается)
BH = атрибут, используемый для очищаемой области
CH = y-координата, верхний левый угол окна
CL = x-координата, верхний левый угол окна
DH = y-координата, нижний правый угол окна
DL = x-координата, нижний правый угол окна

При возврате. Ничего не возвращается.

Замечания. В видеорежимах, поддерживающих несколько страниц, эта функция влияет только на страницу, отображаемую в настоящий момент на экране.

Если содержимое AL не равно 00H, область в указанном окне прокручивается вверх на заданное число строк. Текст, уходящий за верхний срез окна, теряется. Новые строки, появляющиеся внизу окна, заполнены пробелами ASCII с атрибутами, определенными регистром BH.

Для прокрутки содержимого окна вниз используйте прерывание Int 10H с функцией 07H.

Int 10H [MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]
Функция 07H
Инициализировать или прокрутить окно вниз

Инициализирует указанное видеоокно, заполняя его символами пробелов ASCII с заданным атрибутом, либо прокручивает содержимое окна вниз на заданное число строк.

При вызове. AH = 07H
AL = число строк прокрутки (если 0, все окно очищается)
BH = атрибут, используемый для очищаемой области
CH = y-координата, верхний левый угол окна
CL = x-координата, верхний левый угол окна
DH = y-координата, нижний правый угол окна
DL = x-координата, нижний правый угол окна

При возврате. Ничего не возвращается.

Замечания. В видеорежимах, поддерживающих несколько страниц, эта функция влияет только на страницу, отображаемую в настоящий момент на экране.

Если содержимое AL не равно 00H, область в указанном окне прокручивается вниз на заданное число строк. Текст, уходящий за нижний срез окна, теряется. Новые строки, появляющиеся вверху окна, заполнены пробелами ASCII с атрибутами, определенными регистром BH.

Для прокрутки содержимого окна вверх используйте прерывание Int 10H с функцией 06H.

Int 10H [MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]
Функция 08H
Прочитать символ и атрибут в позиции курсора

Возвращает символ ASCII и его атрибут в текущей позиции курсора для указанной видеостраницы.

При вызове. AH = 08H
BH = страница

При возврате. AH = атрибут
AL = символ

Замечание. В видеорежимах, поддерживающих несколько страниц, символ и его атрибут могут быть прочитаны с любой страницы независимо от того, какая страница отображается в настоящий момент на экране.

Int 10H [MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]
Функция 09H
Записать символ и атрибут в позицию курсора

Записывает символ ASCII и его атрибут в текущую позицию курсора на экране.

При вызове. AH = 09H
AL = символ
BH = страница
BL = атрибут (текстовые режимы) или цвет (графические режимы)
CX = число записываемых символов (коэффициент повторения)

При возврате. Ничего не возвращается.

Замечания. В графических режимах коэффициент повторения в CX приводит к правиль-

ному результату только для текущей строки. Если записывается больше символов, чем число оставшихся столбцов в текущей строке, результат непредсказуем.

Все значения в AL так или иначе отображаются на экране; управляющие символы, включая гудок, забой, возврат каретки и перевод строки, не распознаются как специальные символы и не влияют на положение курсора.

После записи символа курсор следует явно переместить в следующую позицию с помощью прерывания Int 10H с функцией 02H.

Для записи символа в текущую позицию курсора его атрибута используйте прерывание Int 10H с функцией 0AH.

Если настоящая функция используется для записи символов в графическом режиме и бит 7 BL установлен (1), будет выполнена операция ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR) записываемого символа с текущим содержимым экрана. Это свойство можно использовать для записи символов и их последующего «стирания».

При использовании CGA и PCjr в графических режимах 04H–06H шаблоны изображений для символов с кодами 80H–FFH берутся из таблицы, адрес которой записан в векторе прерывания Int 1FH. В PCjr адрес таблицы для символов с кодами 00H–7FH записан в векторе прерывания Int 44H. Для установки другого набора символов его следует загрузить в память и исправить содержимое этого вектора.

При использовании EGA, MCGA и VGA в графических режимах адрес таблицы определения символов хранится в векторе прерывания Int 43H. (См. Int 10H с функцией 11H.)

Int 10H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 0AH (10)
Записать символ в позицию курсора

Записывает символ ASCII в текущую позицию курсора на экране. Символ получает атрибут символа, ранее находившегося в этой позиции.

При вызове. AH = 0AH
AL = символ
BH = страница
BL = цвет (графические режимы, только PCjr)
CX = число записываемых символов (коэффициент повторения)

При возврате. Ничего не возвращается.

Замечания. В графических режимах коэффициент повторения в CX приводит к правильному результату только для текущей строки. Если записывается больше символов, чем число оставшихся столбцов в текущей строке, результат непредсказуем.

Все значения в AL так или иначе отображаются на экране; управляющие символы, включая гудок, забой, возврат каретки и перевод строки, не распознаются как специальные символы и не влияют на положение курсора.

После записи символа курсор следует явно переместить в следующую позицию с помощью прерывания Int 10H с функцией 02H.

Для записи символа с атрибутом в текущую позицию курсора используйте прерывание Int 10H с функцией 09H.

Если настоящая функция используется для записи символов в графическом режиме и бит 7 BL установлен (1), будет выполнена операция ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR) записываемого символа с текущим содержимым экрана. Это свойство можно использовать для записи символов и их последующего «стирания».

При использовании CGA и PCjr в графических режимах 04H–06H шаблоны изображений всех символов с кодами 80H–FFH берутся из таблицы, адрес которой записан в векторе прерывания Int 1FH. В PCjr адрес таблицы для символов с кодами

00H–7FH записан в векторе прерывания Int 44H. Для установки другого набора символов его следует загрузить в память и исправить содержимое этого вектора.

При использовании EGA, MCGA и VGA в графических режимах адрес таблицы определения символов хранится в векторе прерывания Int 43H. (См. Int 10H с функцией 11H.)

Int 10H [CGA] [PCjr] [EGA] [MCGA] [VGA]
Функция 0BH (11)
Установить палитру, фон или край экрана.

Выбирает цвета палитры, фона или края экрана.

При вызове. Для установки цвета фона и цвета края экрана в графических режимах или цвета края экрана в текстовых режимах
AH = 0BH
BH = 00H
BL = цвет
Для выбора палитры (4-цветные графические режимы 320 × 200)
AH = 0BH
BH = 01H
BL = палитра (см. *Замечания*)

При возврате. Ничего не возвращается.

Замечания. В текстовых режимах эта функция выбирает только цвет края экрана. Цвет фона каждого отдельного символа определяется старшими 4 бит байта атрибута этого символа.

При использовании CGA и EGA эта функция осуществляет выбор палитры только в 4-цветных графических режимах 320 × 200. В 4-цветных графических режимах 320 × 200, если регистр BH = 01H, можно выбрать следующие палитры:

Палитра	Значение пиксела	Цвет
0	0	Цвет фона
	1	Зеленый
	2	Красный
	3	Коричневый или желтый
1	0	Цвет фона
	1	Бирюзовый
	2	Фиолетовый
	3	Белый

При использовании CGA в двухцветном графическом режиме 640 × 200 цвет фона, выбранный с помощью этой функции, фактически определяет цвет ненулевых пикселов; нулевые пикселы всегда изображаются черным цветом.

При использовании PCjr в двухцветном графическом режиме 640 × 200, если BH = 00H и бит 0 регистра BL очищен, пиксел со значением 1 отображается белым; если бит 0 установлен, пиксел со значением 1 отображается черным.

См. также Int 10H с функцией 10H, которая используется для программирования палитры PCjr, EGA, MCGA и VGA.

Int 10H [CGA] [PCjr] [EGA] [MCGA] [VGA]
Функция 0CH (12)
Записать графический пиксел

Рисует на экране точку с указанными графическими координатами.

При вызове. AH = 0CH
AL = значение пиксела
BH = страница
CX = столбец (графическая x-координата)
CL = строка (графическая y-координата)
При возврате. Ничего не возвращается.
Замечания. Диапазон допустимых значений пикселов и (x, y)-координат зависит от текущего видеорежима.

Если бит 7 в AL установлен, новое значение пиксела будет соответствовать результату операции ИСКЛЮЧАЮЩЕЕ ИЛИ (XOR) с текущим содержимым пиксела.

В видеорежимах, поддерживающих только одну страницу, регистр BH игнорируется.

Int 10H **[CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 0DH (13)
Прочитать графический пиксел

Возвращает текущее значение пиксела в точку экрана с указанными графическими координатами.
При вызове. AH = 0DH
BH = страница
CX = столбец (графическая x-координата)
DX = строка (графическая y-координата)
При возврате. AL = значение пиксела
Замечания. Диапазон допустимых (x, y)-координат и возможные значения пикселов определяются текущим видеорежимом.

В видеорежимах, поддерживающих только одну страницу, регистр BH игнорируется.

Int 10H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 0EH (14)
Записать символ в режиме телетайпа

Записывает символ ASCII в текущую позицию курсора на экране с использованием указанного цвета (при работе в графических режимах), после чего выполняет соответствующий инкремент позиции курсора.
При вызове. AH = 0EH
AL = символ
BH = страница
BL = цвет изображения (графические режимы)
При возврате. Ничего не возвращается.
Замечания. Специальные коды ASCII для гудка (07H), забоя (08H), возврата каретки (0DH) и перевода строки (0AH) распознаются, и выполняются соответствующие им действия. Все остальные символы записываются на экран (даже если они являются управляющими символами), и курсор перемещается в следующую позицию.

В видеорежимах, поддерживающих несколько страниц, символы могут записываться на любую страницу независимо от того, какая страница отображается в настоящий момент на экране.

Выполняются переход на новую строку и прокрутка строк. Если курсор находится в конце строки, он перемещается в начало следующей. Если курсор достигает конца последней строки на экране, изображение прокручивается на одну строку и курсор

перемещается в начало новой пустой строки. Атрибут для всей новой строки принимается таким же, как у последнего символа, записанного на предыдущую строку.

Стандартный драйвер консоли MS-DOS (CON) записывает текст на экран с помощью этой функции. Функцию нельзя использовать для спецификации атрибута символа. Для того чтобы записать на экран символ с заданным атрибутом, можно сначала в текущую позицию курсора записать пробел ASCII (20H) с требуемым атрибутом с помощью прерывания Int 10H с функцией 09H, а затем записать сам символ с помощью прерывания Int 10H с функцией 0EH. Этот способ, хотя и не отличается изяществом, избавляет программу от необходимости в явной форме организовывать смену строк и прокрутку изображения.

См. также Int 10H с функцией 13H.

Int 10H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 0FH (15)
Получить видеорежим

Возвращает текущий видеорежим активного видеоконтроллера.
При вызове. AH = 0FH
При возврате. AH = число столбцов символов на экране
AL = видеорежим (см. Int 10H с функцией 00H)
BH = активная видеостраница
Замечание. Эту функцию можно использовать для получения ширины экрана перед его очисткой с помощью прерывания Int 10H с функциями 06H и 07H.

Int 10H **[PCjr] [EGA] [MCGA] [VGA]**
Функция 10H (16), подфункция 00H
Установить регистр палитры

Устанавливает соответствие регистра палитры отображаемому цвету.
При вызове. С адаптерами PCjr, EGa и VGA
AH = 10H
AL = 00H
BH = значение цвета
BL = регистр палитры (00–0FH)
С адаптером MCGA
AH = 10H
AL = 00H
BX = 0712H
При возврате. Ничего не возвращается.
Замечание. При использовании видеоплаты MCGA эта функция может вызываться только с BX = 0712H и выбирать набор регистров с последовательностью 8 цветов.

Int 10H **[PCjr] [EGA] [VGA]**
Функция 10H (16), подфункция 01H
Установить цвет края экрана

Управляет цветом края экрана (выбегов развертки).
При вызове. AH = 10H
AL = 01H
BH = значение цвета
При возврате. Ничего не возвращается.

Int 10H **[PCjr] [EGA] [VGA]**
Функция 10H (16), подфункция 02H
Установить палитру и край экрана

Устанавливает все регистры палитры и цвет края экрана (выбегов развертки) в одной операции.
При вызове. AH = 10H
AL = 02H
ES:DX = сегмент:относительный_адрес списка цветов
При возврате. Ничего не возвращается.
Замечания. Список цветов имеет длину 17 байт. Первые 16 байт составляют значения цветов, загружаемых в регистры 0–15 палитры, а последний байт хранится в регистре цвета края экрана.

В 16-цветных графических режимах по умолчанию устанавливается следующая палитра:

Значение пиксела	Цвет
01H	Синий (blue)
02H	Зеленый (green)
03H	Бирюзовый (cyan)
04H	Красный (red)
05H	Филетовый (magenta)
06H	Коричневый (brown)
07H	Белый (white)
08H	Серый (gray)
09H	Голубой (light blue)
0AH	Салатовый (light green)
0BH	Светло-бирюзовый (light cyan)
0CH	Светло-красный (light red)
0DH	Светло-фиолетовый (light magenta)
0EH	Желтый (yellow)
0FH	Ярко-белый (intense white)

Int 10H **[PCjr] [EGA] [MCGA] [VGA]**
Функция 10H (16), подфункция 03H
Переключить бит мерцание/яркость

Определяет назначение старшего бита атрибута символа: выбор мерцания или повышенной яркости.
При вызове. AH = 10H
AL = 03H
BL = переключатель мерцание/яркость
0 = включить яркость
1 = вкдючить мерцание
При возврате. Ничего не возвращается.

Int 10H **[VGA]**
Функция 10H (16), подфункция 07H
Получить регистр палитры

Возвращает цвет, связанный с указанным регистром палитры.
При вызове. AH = 10H

AL = 07H
BL = регистр палитры
При возврате. BH = цвет

Int 10H [VGA]
Фунция 10H (16), подфункция 08H
Получить цвет края экрана

Возвращает текущий цвет края экрана (выбегов развертки)
При вызове. AH = 10H
AL = 08H
При возврате. BH = цвет

Int 10H [VGA]
Функция 10H (16), подфункция 09H
Получить палитру и край экрана

Возвращает содержимое регистров палитры и цвет края экрана (выбегов развертки) в одной операции.
При вызове. AH = 10H
AL = 09H
ES:DX = сегмент:относительный_адрес 17-байт буфера
При возврате. ES:DX = сегмент:относительный_адрес буфера, причем буфер содержит значения палитры в байтах 00H–0FH и цвет края экрана в байте 10H.

Int 10H [MCGA] [VGA]
Функция 10H (16), подфункция 10H (16)
Установить регистр цвета

Программирует индивидуальный регистр цвета комбинацией красного–зеленого–синего (RGB).
При вызове. AH = 10H
AL = 10H
BX = регистр цвета
CH = значение зеленого
CL = значение синего
DH = значение красного
При возврате. Ничего не возвращается.
Замечание. Если разрешено определение яркости полутонового изображения, взвешенное значение яркости вычисляется по правилам, данным в описании Int 10H с функцией 10H, подфункцией 1BH, и записывается во все три компонента регистра цвета. (См. также Int 10H с функцией 12H, подфункцией 33H.)

Int 10H [MCGA] [VGA]
Функция 10H (16), подфункция 12H (18)
Установить блок регистров цветов

В одной операции программирует группу последовательных регистров цветов.
При вызове. AH = 10H
AL = 12H

BX = первый регистр цвета
CX = число регистров цветов
ES:DX = сегмент:относительный_адрес цветовой таблицы

При возврате. Ничего не возвращается.

Замечания. Таблица состоит из последовательности 3-байт записей, по одной записи на каждый программируемый регистр цвета. Байты каждой записи определяют значения красного, зеленого и синего (в указанном порядке) для соответствующего регистра цвета. Если разрешено определение яркости полутонового изображения, взвешенное значение яркости для каждого регистра вычисляется по правилам, данным в описании Int 10H с функцией 10H, подфункцией 1BH, и записывается во все три компонента регистра цвета. (См. также Int 10H с функцией 12H, подфункцией 33H.)

Int 10H **[VGA]**
Функция 10H (16), подфункция 13H (19)
Установить состояние цветовой страницы

Выбирает страничный режим регистров цветов либо конкретную страницу регистров цветов.

При вызове. Для выбора страничного режима
AH = 10H
AL = 13H
BH = страничный режим
00H 4 страницы по 64 регистра
01H 16 страниц по 16 регистров
BL = 00H
Для выбора страницы регистров цветов
AH = 10H
AL = 13H
BH = страница
BL = 01H

При возврате. Ничего не возвращается.

Замечание. Эта функция не работает в режиме 13H (320 × 200, 256 цветов, графика).

Int 10H **[MCGA] [VGA]**
Функция 10H (16), подфункция 15H (21)
Получить регистр цвета

Возвращает содержимое регистра цвета в виде его красного, зеленого и синего компонентов.

При вызове. AH = 10H
AL = 15H
BX = регистр цвета

При возврате. CH = значение зеленого
CL = значение синего
DH = значение красного

Int 10H **[MCGA] [VGA]**
Функция 10H (16), подфункция 17H (23)
Получить блок регистров цветов

Позволяет в одной операции прочитать красный, зеленый и синий компоненты каждого регистра из набора регистров цветов.

При вызове. AH = 10H
AL = 17H
BX = первый регистр цвета
CX = число регистров цветов
ES:DX = сегмент:относительный_адрес буфера для приема списка цветов
При возврате. ES:DX = сегмент:относительный_адрес буфера, причем буфер содержит список цветов
Замечание. Список цветов, возвращенный в буфер пользователя, состоит из последовательности 3-байт записей, соответствующих регистрам цветов. Каждая 3-байт запись содержит красный, зеленый и синий компоненты регистра в указанном порядке.

Int 10H **[VGA]**
Функция 10H (16), подфункция 1AH (26)
Получить состояние цветовой страницы

Возвращает страничный режим регистров цветов и текущую цветовую страницу.
При вызове. AH = 10H
AL = 1AH
При возврате. BH = цветовая страница
BL = страничный режим
00H 4 страницы по 64 регистра
01H 16 страниц по 16 регистров
Замечание. См. Int 10H с функцией 10H и подфункцией 13H – выбор страничного режима или текущей цветовой страницы.

Int 10H **[MCGA] [VGA]**
Функция 10H (16), подфункция 1BH (27)
Установить значения яркости полутонового изображения

Преобразует значения красного, зеленого и синего одного или нескольких регистров цветов в эквивалентные значения яркости полутонового изображения.
При вызове. AH = 10H
AL = 1BH
BX = первый регистр цвета
CX = число регистров цветов
При возврате. Ничего не возвращается.
Замечание. Для каждого регистра цвета вычисляется взвешенная сумма его значений красного, зеленого и синего (30% красного + 59% зеленого + 11% синего), которая затем записывается назад во все три компонента регистра цвета. Первоначальные значения красного, зеленого и синего теряются.

Int 10H **[EGA] [MCGA] [VGA]**
Функция 11H (17), подфункции 00H и 10H (16)
Загрузить шрифт пользователя

Загружает в указанный блок ОЗУ генератора символов таблицу (определение символов) шрифта пользователя.
При вызове. AH = 11H
AL = 00H или 10H (см. *Замечания*)

BH = пункты [1] (байтов на символ)
BL = блок
CX = число символов, определенных таблицей
DX = первый код символа в таблице
ES:BP = сегмент:относительный_адрес таблицы шрифта

При возврате. Ничего не возвращается.

Замечания. Функция обеспечивает выбор шрифта в текстовых (алфавитно-цифровых) видеорежимах. Для выбора шрифта в графических (с адресацией ко всем точкам) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 20H–24H.

Если AL = 10H, активной должна быть страница 0. Пересчитываются пункты (байтов на символ) и строки, а также длина буфера регенерации. Контроллер перепрограммируется на максимальное число строк развертки (пункты −1), начало курсора (пункты −2), конец курсора (пункты −1), конец экрана по вертикали ((число строк * пункты) −1), а также позицию подчеркивания (пункты −1, только в режиме 7).

Если подфункция 10H вызывается не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Для видеоплаты MCGA за вызовом подфункции 00H должен следовать вызов подфункции 03H, чтобы программа ПЗУ BIOS загрузила шрифт во внутренние шрифтовые страницы генератора символов.

Для видеоплаты MCGA подфункция 10H зарезервирована. При ее вызове выполняется подфункция 00H.

Int 10H **[EGA] [VGA]**
Функция 11H(17), подфункции 01H и 11H(17)
Загрузить шрифт ПЗУ 8 × 14

Загружает используемый по умолчанию шрифт 8 × 14 из ПЗУ BIOS в указанный блок ОЗУ генератора символов.

При вызове. AH = 11H
AL = 01H или 11H (см. *Замечания*)
BL = блок

При возврате. Ничего не возвращается.

Замечания. Функция обеспечивает выбор шрифта в текстовых (алфавитно-цифровых) видеорежимах. Для выбора шрифта в графических (с адресацией ко всем точкам) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 20H–24H.

Если AL = 11H, активной должна быть страница 0. Пересчитываются пункты (байтов на символ) и строки, а также длина буфера регенерации. Контроллер перепрограммируется на максимальное число строк развертки (пункты −1), начало курсора (пункты −2), конец курсора (пункты −1), конец экрана по вертикали ((число строк × пункты) −1), а также позицию подчеркивания (пункты −1, только в режиме 7).

Если подфункция 10H называется не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Для видеоплаты MCGA подфункции 01H и 11H зарезервированы. При вызове любой из них выполняется подфункция 04H.

---

[1] Слово points, которое мы сочли возможным перевести типографским термином «пункт» (единица длины, используемая для определения размеров шрифтов), обозначает здесь число точек (пикселов) по вертикали в матрице символа. Поскольку ширина символа составляет 8 пикселов (1 байт таблицы символов), то высота символа в «пунктах» совпадает с числом байтов, занимаемым определением символа в таблице символов. – *Прим. перев.*

Int 10H [EGA] [MCGA] [VGA]
Функция 11H(17), подфункции 02H и 12H(18)
Загрузить шрифт ПЗУ 8 × 8

Загружает используемый по умолчанию шрифт 8 × 8 из ПЗУ BIOS в указанный блок ОЗУ генератора символов.

При вызове. AH = 11H
AL = 02H или 12H (см. *Замечания*)
BL = блок

При возврате. Ничего не возвращается.

Замечания. Функция обеспечивает выбор шрифта в текстовых (алфавитно-цифровых) видеорежимах. Для выбора шрифта в графических (с адресацией ко всем точкам) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 20H–24H.

Если AL = 12H, страница 0 должна быть активной. Пересчитываются пункты (байтов на символ) и строки, а также длина буфера регенерации. Контроллер перепрограммируется на максимальное число строк развертки (пункты −1), начало курсора (пункты −2), конец курсора (пункты −1), конец экрана по вертикали ((число строк * пункты) −1), а также позицию подчеркивания (пункты −1, только в режиме 7).

Если подфункция 11H вызывается не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Для видеоплаты MCGA за вызовом подфункции 02H должен следовать вызов подфункции 03H, чтобы программа ПЗУ BIOS загрузила шрифт во внутренние шрифтовые страницы генератора символов. Для видеоплаты MCGA подфункция 12H зарезервирована. При ее вызове выполняется подфункция 02H.

Int 10H [EGA] [MCGA] [VGA]
Функция 11H(17), подфункция 03H
Установить описатель блоков

Определяет блоки символов, выбираемые битом 3 байта атрибута символа в алфавитно-цифровых (текстовых) режимах.

При вызове. AH = 11H
AL = 03H
BL = код выбора блока генератора символов (см. *Замечания*)

При возврате. Ничего не возвращается.

Замечания. С видеоплатами EGA и MCGA биты регистра BL используются следующим образом:

Биты	Значение
0–1	блок символов, выбираемый байтами атрибута, у которых бит 3 = 0
2–3	блок символов, выбираемый байтами атрибута, у которых бит 3 = 1
4–7	не используются (должны содержать 0)

С видеоплатой VGA биты регистра BL используются следующим образом:

Биты	Значение
0, 1, 4	блок символов, выбираемый байтами атрибута, у которых бит 3 = 0
2, 3, 5	блок символов, выбираемый байтами атрибута, у которых бит 3 = 1
6, 7	не используются (должны содержать 0)

При использовании 256-символьного набора оба поля регистра BL должны выбирать один и тот же блок символов. В этих случаях бит 3 атрибута символа управляет яркостью изображения. При использовании 512-символьных наборов поля регистра BL определяют блоки, каждый из которых содержит половину набора символов, а бит 3 атрибута символа выбирает старшую или младшую половину набора символов.

При использовании 512-символьного набора рекомендуется вызывать Int 10H с функцией 10H и подфункцией 00H при BX = 0712H, чтобы установить набор регистров с последовательностью 8 цветов.

Int 10H **[MCGA] [VGA]**
Функция 11H (17), подфункции 04H и 14H (20)
Загрузить шрифт ПЗУ 8 × 16

Загружает используемый по умолчанию шрифт 8 × 16 из ПЗУ BIOS в указанный блок ОЗУ генератора символов.

При вызове. AH = 11H
AL = 04H или 14H (см. *Замечания*)
BL = блок

При возврате. Ничего не возвращает.

Замечания. Функция обеспечивает выбор шрифта в текстовых (алфавитно-цифровых) видеорежимах. Для выбора шрифта в графических (с адресацией ко всем точкам) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 20H–24H.

Если AL = 14H, страница 0 должна быть активной. Пересчитываются пункты (байтов на символ) и строки, а также длина буфера регенерации. Контроллер перепрограммируется на максимальное число строк развертки (пункты −1), начало курсора (пункты −2), конец курсора (пункты −1), конец экрана по вертикали (число строк * пункты −1 в режимах, использующих 350 и 400 строк развертки, или число строк * пункты * 2−1 в режимах, использующих 200 строк развертки), а также позицию подчеркивания (пункты −1, только в режиме 7).

Если подфункция 14H вызывается не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Для видеоплаты MCGA за вызовом подфункции 04H должен следовать вызов подфункции 03H, чтобы программа ПЗУ BIOS загрузила шрифт во внутренние шрифтовые страницы генератора символов.

Для видеоплаты MCGA подфункция 14H зарезервирована. При ее вызове выполняется подфункция 04H.

Int 10H **[EGA] [MCGA] [VGA]**
Функция 11H (17), подфункция 20H (32)
Установить указатель шрифта Int 1FH

Устанавливает указатель Int 1FH на таблицу шрифта пользователя. Эта таблица используется для кодов символов 80H–FFH в графических режимах 04H–06H.

При вызове. AH = 11H
AL = 20H
ES:BP = сегмент:относительный_адрес таблицы шрифта

При возврате. Ничего не возвращается.

Замечания. Функция обеспечивает выбор шрифта в графических (с адресацией ко всем точкам) видеорежимах. Для выбора шрифта в текстовых (алфавитно-цифровых) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 00H–14H.

Если эта подфункция вызывается не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Int 10H **[EGA] [MCGA] [VGA]**
Функция 11H (17), подфункция 21H (33)
Настроить Int 43H на шрифт пользователя

Устанавливает вектор прерывания Int 43H так, чтобы он указывал на таблицу шрифта пользователя, и обновляет область видеоданных ПЗУ BIOS. Видеоконтроллер не перепрограммируется.

При вызове. AH = 11H
AL = 21H
BL = описатель числа строк символов 00H, если указано пользователем (см. регистр DL)
01H = 11 (0EH) строк
02H = 25 (19H) строк
03H = 43 (2BH) строк
CX = пункты (байтов на символ)
DL = число строк символов на экран (если BL = 00H)
ES : BP = сегмент : относительный_адрес таблицы шрифта пользователя

При возврате. Ничего не возвращается.

Замечания. Функция обеспечивает выбор шрифта в графических (с адресацией ко всем точкам) видеорежимах. Для выбора шрифта в текстовых (алфавитно-цифровых) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 00H–14H.

Если эта подфункция вызывается не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Int 10H **[EGA] [MCGA] [VGA]**
Функция 11H (17), подфункция 22H (34)
Настроить Int 43H на шрифт ПЗУ 8 × 14

Устанавливает вектор прерывания Int 43H так, чтобы он указывал на используемый по умолчанию шрифт ПЗУ BIOS 8 × 14, и обновляет область видеоданных ПЗУ BIOS. Видеоконтроллер не перепрограммируется.

При вызове. AH = 11H
AL = 22H
BL = описатель числа строк символов
00H, если указано пользователем (см. регистр DL)
01H = 14 (0EH) строк
02H = 25 (19H) строк
03H = 43 (2BH) строк
DL = число строк символов на экран (если BL = 00H)

При возврате. Ничего не возвращается.

Замечания. Функция обеспечивает выбор шрифта в графических (с адресацией ко всем точкам) видеорежимах. Для выбора шрифта в текстовых (алфавитно-цифровых) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 00H–14H.

Если эта подфункция вызывается не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Для видеоплаты MCGA вызов этой подфункции приводит к выполнению подфункции 24H.

Int 10H [EGA] [MCGA] [VGA]
Функция 11H (17), подфункция 23H (35)
Настроить Int 43H на шрифт ПЗУ 8 × 8

Устанавливает вектор прерывания Int 43H так, чтобы он указывал на используемый по умолчанию шрифт ПЗУ BIOS 8 × 8, и обновляет область видеоданных ПЗУ BIOS. Видеоконтроллер не перепрограммируется.
При вызове. AH = 11H
AL = 23H
BL = описатель числа строк символов
00H, если указано пользователем (см. регистр DL)
01H = 14 (0EH) строк
02H = 25 (19H) строк
03H = 43 (2BH) строк
DL = число строк символов на экран (если BL = 00H)
При возврате. Ничего не возвращается.
Замечания. Функция обеспечивает выбор шрифта в графических (с адресацией ко всем точкам) видеорежимах. Для выбора шрифта в текстовых (алфавитно-цифровых) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 00H–14H.

Если эта подфункция вызывается не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Int 10H [MCGA] [VGA]
Функция 11H (17), подфункция 24H (36)
Настроить Int 43H на шрифт ПЗУ 8 × 16

Настраивает вектор прерывания Int 43H так, чтобы он указывал на используемый по умолчанию шрифт ПЗУ BIOS 8 × 16, и обновляет область видеоданных ПЗУ BIOS. Видеоконтроллер не перепрограммируется.
При вызове. AH = 11H
AL = 24H
BL = описатель числа строк символов
00H, если указано пользователем (см. регистр DL)
01H = 14 (0EH) строк
02H = 25 (19H) строк
03H = 43 (2BH) строк
DL = число строк символов на экран (если BL = 00H)
При возврате. Ничего не возвращается.
Замечания. Функция обеспечивает выбор шрифта в графических (с адресацией ко всем точкам) видеорежимах. Для выбора шрифтов в текстовых (алфавитно-цифровых) режимах используйте прерывание Int 10H с функцией 11H и подфункциями 00H–14H.

Если эта подфункция вызывается не непосредственно после установки режима, а в любой другой момент, то результаты непредсказуемы.

Int 10H [EGA] [MCGA] [VGA]
Функция 11H (17), подфункция 30H (48)
Получить информацию о шрифте

Возвращает указатель таблицы определения символов для шрифта, а также пункты (байтов на символ) и число строк для этого шрифта.
При вызове. AH = 11H

AL = 30H
BH = код шрифта
00H = текущее содержимое Int 1FH
01H = текущее содержимое Int 43H
02H = шрифт ПЗУ 8 × 14 (только EGA, VGA)
03H = шрифт ПЗУ 8 × 8 (символы 00H–7FH)
04H = шрифт ПЗУ 8 × 8 (символы 80H–FFH)
05H = дополнительный шрифт ПЗУ 9 × 14 (только EGA, VGA)
06H = шрифт ПЗУ 8 × 16 (только MCGA, VGA)
07H = дополнительный шрифт ПЗУ 9 × 16 (только VGA)

При возврате. CX = пункты (байтов на символ)
DL = число строк (число строк символов на экране −1)
ES:BP = сегмент:относительный_адрес таблицы шрифта

Int 10H **[EGA] [VGA]**
Функция 12H(18), подфункция 10H(16)
Получить информацию о конфигурации

Обеспечивает получение информации о конфигурации для активной видеоподсистемы.

При вызове. AH = 12H
AL = 10H

При возврате. BH = тип дисплея
0, если дисплей цветной
1, если дисплей монохромный
BL = память, установленная на плате
00H, если установлено 64 Кбайт
01H, если установлено 128 Кбайт
02H, если установлено 192 Кбайт
03H, если установлено 256 Кбайт
CH = биты характеристик (см. *Замечания*)
CL = настройка переключателя (см. *Замечания*)

Замечания. Биты характеристик устанавливаются по содержимому регистра 0 состояния ввода и в ответ на состояние указанных битов регистра управления характеристиками:

Бит(ы) характеристик	Бит управления характеристиками	Бит состояния ввода
0	0	5
1	0	6
2	1	5
3	1	6
4–7	Не используются	

Биты в байте настройки переключателя указывают на состояние микропереключателя конфигурации EGA (1 = выкл, 0 = вкл):

Бит(ы)	Значение
0	переключатель конфигурации 1
1	переключатель конфигурации 2
2	переключатель конфигурации 3
3	переключатель конфигурации 4
4–7	не используются

Int 10H **[EGA] [VGA]**
Функция 12H(18), подфункция 20H(32)
Выбрать дополнительную программу печати экрана

Выбирает дополнительную программу печати экрана для видеоплат EGA и VGA, правильно функционирующую при длине экрана, отличающейся от 25 строк.
При вызове. AH = 12H
BL = 20H
При возврате. Ничего не возвращается.

Int 10H **[VGA]**
Функция 12H(18), подфункция 30H(48)
Установить число строк развертки

Выбирает число строк развертки для алфавитно-цифровых режимов. Выбранное значение вступает в силу при следующем вызове Int 10H с функцией 00H выбора видеорежима.
При вызове. AH = 12H
AL = код числа строк развертки
00H = 200 строк развертки
01H = 350 строк развертки
02H = 400 строк развертки
BL = 30H
При возврате. Если видеоплата EGA активна
AL = 12H
Если видеоплата EGA неактивна
AL = 00H

Int 10H **[MCGA] [VGA]**
Функция 12H(18), подфункция 31H(49)
Разрешить/запретить загрузку используемой по умолчанию палитры

Разрешает или запрещает загрузку используемой по умолчанию палитры при выборе видеорежима.
При вызове. AH = 12H
AL = 00H для разрешения загрузки используемой по умолчанию палитры
01H для запрещения загрузки используемой по умолчанию палитры
При возврате. Если функция поддерживается
AL = 12H

Int 10H **[MCGA] [VGA]**
Функция 12H(18), подфункция 32H(50)
Разрешить/запретить функционирование видеосистемы

Разрешает или запрещает доступ ЦП к портам ввода-вывода видеоадаптеров и буферу видеорегенерации.
При вызове. AH = 12H
AL = 00H для разрешения доступа
01H для запрещения доступа
BL = 32H
При возврате. Если функция поддерживается
AL = 12H

Int 10H [MCGA] [VGA]
Функция 12H(18), подфункция 33H(51)
Разрешить/запретить определение яркости полутонового изображения

Разрешает или запрещает определение яркости активного в настоящий момент полутонового изображения.

При вызове. AH = 12H
AL = 00H для разрешения определения яркости полутонового изображения
01H для запрещения определения яркости полутонового изображения
BL = 33H

При возврате. Если функция поддерживается
AL = 12H

Замечание. Если определение яркости полутонового изображения разрешено, оно выполняется в процессе выбора видеорежима, программирования палитры и загрузки регистра цвета.

Int 10H [VGA]
Функция 12H(18), подфункция 34H(52)
Разрешить/запретить эмуляцию курсора

Разрешает или запрещает эмуляцию курсора для активного в настоящий момент изображения. Если эмуляция курсора включена, ПЗУ автоматически переотображает установленные с помощью прерывания Int 10H с функцией 01H значения начальной и конечной строк развертки для курсора на текущие размеры ячейки символа.

При вызове. AH = 12H
AL = 00H для разрешения эмуляции курсора
01H для запрещения эмуляции курсора
BL = 34H

При возврате. Если функция поддерживается
AL = 12H

Int 10H [MCGA] [VGA]
Функция 12H(18), подфункция 35H(53)
Переключить активную видеосистему

Разрешает выбор одного из двух входящих в систему видеоадаптеров, если использование памяти или адресов портов двумя адаптерами вступает в противоречие.

При вызове. AH = 12H
AL = функция переключения
00H для запрещения функционирования начального видеоадаптера
01H для разрешения функционирования видеоадаптера на системной плате
02H для запрещения функционирования активного видеоадаптера
03H для разрешения функционирования неактивного видеоадаптера
BL = 35H
ES:DX = сегмент:относительный_адрес 128-байт буфера (если AL = = 00H, 02H или 03H)

При возврате. Если функция поддерживается
AL = 12H
и, если при вызове AL = 00H или 02H

Информация о состоянии видеоадаптера сохраняется в буфере вызывающей задачи
или, если при вызове AL = 03H
Состояние видеоадаптера восстанавливается по информации, хранящейся в буфере вызывающей задачи

Замечания. Эту подфункцию можно использовать, только если оба видеоадаптера поддерживают функцию запрещения функционирования (Int 10H с функцией 12H и подфункцией 32H).

Если видеосистема на системной плате и видеосистема на адаптерной плате при использовании памяти или портов не вступают в противоречие, оба видеоконтроллера могут быть активными одновременно и данная подфункция не требуется.

Int 10H **[VGA]**
Функция 12H(18), подфункция 36H(54)
Разрешить/запретить регенерацию экрана

Разрешает или запрещает видеорегенерацию активной в настоящий момент видеосистемы.

При вызове. AH = 12H
AL = 00H для разрешения регенерации
01H для запрещения регенерации
BL = 36H

При возврате. Если функция поддерживается
AL = 12H

Int 10H **[MDA] [CGA] [PCjr] [EGA] [MCGA] [VGA]**
Функция 13H(19)
Записать строку в режиме телетайпа

Пересылает строку в видеобуфер активной в настоящей момент видеосистемы, начиная с заданной позиции.

При вызове. AH = 13H
AL = режим записи
0 атрибут в BL; строка содержит только коды символов; позиция курсора после записи не обновляется
1 атрибут в BL, строка содержит только коды символов; позиция курсора обновляется после записи
2 строка содержит попеременно коды символов и байты атрибута; позиция курсора после записи не обновляется
3 строка содержит попеременно коды символов и байты атрибута; позиция курсора обновляется после записи
BH = страница
BL = атрибут, если AL = 00H или 01H
CX = длина строки символов
DH = y-координата (строка)
DL = x-координата (столбец)
ES:BP = сегмент:относительный_адрес строки

При возврате. Ничего не возвращается.

Замечания. Эта функция доступна на исходных компьютерах IBM PC или PC/XT, только если установлен видеоадаптер EGA (содержащий собственное ПЗУ BIOS).

Функцию можно рассматривать как расширение Int 10H с функцией 0EH. Управ-

ляющие символы: гудок (07H), забой (08H), перевод строки (0AH) и возврат каретки (0DH) распознаются и обрабатываются соответственно.

Int 10H **[PS/2]**
Функция 1AH(26)
Получить или установить код конфигурации видеосистемы

Возвращает код, описывающий установленный видеоадаптер(ы), или обновляет переменную ПЗУ BIOS, описывающую установленный адаптер(ы).
При вызове. AH = 1AH
AL = подфункция
00H = получить код конфигурации видеосистемы
01H = установить код конфигурации видеосистемы
BH = код неактивной системы (если AL = 01H)
BL = код активной системы (если AL = 01H)
При возврате. Если функция поддерживается
AL = 1AH
и, если при вызове AL = 00H
BH = код неактивной видеосистемы
BL = код активной видеосистемы
Замечания. Коды видеосистемы интерпретируются следующим образом:

Код(ы)	Тип видеоподсистемы
00H	Система отсутствует
01H	MDA с монитором 5151
02H	CGA с монитором 5153 или 5154
03H	Зарезервирован
04H	EGA с монитором 5153 или 5154
05H	EGA с монитором 5151
06H	EGA с монитором 5175
07H	VGA с аналоговым монохромным монитором
08H	VGA с аналоговым цветным монитором
09H	Зарезервирован
0AH	MCGA с цифровым цветным монитором
0BH	MCGA с аналоговым монохромным монитором
0CH	MCGA с аналоговым цветным монитором
0DH–FEH	Зарезервированы
FFH	Неизвестная конфигурация

Int 10H **[PS/2]**
Функция 1BH(27)
Получить информацию о функциональной поддержке/состоянии видеосистемы

Позволяет получить информацию о текущем видеорежиме, а также указатель таблицы, описывающей характеристики и возможности видеоадаптера и монитора
При вызове. AH = 1BH
BX = тип реализации (всегда 00H)
ES:DI = сегмент:относительный_адрес 64-байт буфера
При возврате. Если функция поддерживается
AL = 1BH

кроме того, информация помещается в буфер вызывающей задачи (см. *Замечания*)

Замечания. Буфер вызывающей задачи заполняется информацией, содержание которой зависит от текущего видеорежима:

Байт (ы)	Содержимое
00H–03H	Указатель информации о функциональной поддержке (см. следующее *Замечание*)
04H	Текущий видеорежим
05H–06H	Число символьных столбцов
07H–08H	Длина видеобуфера регенерации (байты)
09H–0AH	Начальный адрес верхнего левого угла экрана в буфере
0BH–1AH	Позиция курсора на видеостраницах 0–7 в виде 8 2-байт записей, в которых первый байт является у-координатой, а второй – х-координатой
1BH	Строка развертки начала курсора
1CH	Строка развертки конца курсора
1DH	Активная видеостраница
1EH–1FH	Базовый адрес порта адаптера (3BXH для монохромной системы, 3DXH для цветной)
20H	Текущая установка регистра 3B8H или 3D8H
21H	Текущая установка регистра 3B9H или 3D9H
22H	Число символьных строк
23H–24H	Высота символа в строках развертки
25H	Код активной видеосистемы (см. Int 10H с функцией 1AH)
26H	Код неактивной видеосистемы (см. Int 10H с функцией 1AH)
27H–28H	Число изображаемых цветов (0 для монохромной системы)
29H	Число строк развертки 00H = 200 строк развертки 01H = 350 строк развертки 02H = 400 строк развертки 03H = 480 строк развертки 04H–FFH = зарезервированы
2BH	Первый блок символов (см. Int 10H с функцией 11H и подфункцией 03H)
2CH	Второй блок символов
2DH	Различная информация о состоянии

Бит (ы)	Значение
0	= 1, если все режимы на всех видеосистемах активны (с платой MCGA всегда 0)
1	= 1, если активно определение яркости полутонового изображения
2	= 1, если подключен монохромный монитор
3	= 1, если запрещена загрузка палитры, используемой по умолчанию
4	= 1, если активна эмуляция курсора (с платой MCGA всегда 0)
5	= состояние переключателя яркость/мерцание (0 = яркость, 1 = мерцание)
6–7	зарезервированы

2EH–30H	Зарезервированы
31H	Имеется видеопамять 00H = 64 Кбайт

01H = 128 Кбайт
02H = 192 Кбайт
03H = 256 Кбайт

32H Информация о состоянии указателя сохранения

Бит (ы)	Значение
0	= 1, если активен 512-символьный набор
1	= 1, если активна область динамического сохранения
2	= 1, если активна замена алфавитного шрифта
3	= 1, если активна замена графического шрифта
4	= 1, если активна замена палитры
5	= 1, если активно расширение кода конфигурации системы (DCC)
6–7	зарезервированы

33H–3FH Зарезервированы

Байты 0–3 буфера вызывающей задачи содержат двухсловный указатель (относительный адрес в младшем слове, сегмент в старшем) следующей информации о видеоадаптере и мониторе:

Байт (ы) Содержимое

00H Поддержка видеорежимов

Бит	Значение
0	= 1, если поддерживается режим 00H
1	= 1, если поддерживается режим 01H
2	= 1, если поддерживается режим 02H
3	= 1, если поддерживается режим 03H
4	= 1, если поддерживается режим 04H
5	= 1, если поддерживается режим 05H
6	= 1, если поддерживается режим 06H
7	= 1, если поддерживается режим 07H

01H Поддержка видеорежимов

Бит	Значение
0	= 1, если поддерживается режим 08H
1	= 1, если поддерживается режим 09H
2	= 1, если поддерживается режим 0AH
3	= 1, если поддерживается режим 0BH
4	= 1, если поддерживается режим 0CH
5	= 1, если поддерживается режим 0DH
6	= 1, если поддерживается режим 0EH
7	= 1, если поддерживается режим 0FH

02H Поддержка видеорежимов

Бит (ы)	Значение
0	= 1, если поддерживается режим 10H
1	= 1, если поддерживается режим 11H
2	= 1, если поддерживается режим 12H
3	= 1, если поддерживается режим 13H
4–7	зарезервированы

03H–06H Зарезервированы

07H Число строк развертки в текстовых режимах

Бит (ы)	Значение
0	= 1, если 200 строк развертки
1	= 1, если 350 строк развертки
2	= 1, если 400 строк развертки
3–7	зарезервированы

08H Блоки символов, доступные в текстовых режимах (см. Int 10H с функцией 11H)

09H Максимальное число активных блоков символов в текстовых режимах

0AH Различные возможности BIOS

Бит	Значение
0	= 1, если во всех видеосистемах активны все режимы (с платой MCGA всегда 0)
1	= 1, если возможно определение яркости полутонового изображения
2	= 1, если возможна загрузка символьного шрифта
3	= 1, если возможна загрузка палитры, используемой по умолчанию
4	= 1, если возможна эмуляция курсора
5	= 1, если возможна палитра EGA (64 цвета)
6	= 1, если возможна загрузка регистра цвета
7	= 1, если возможен выбор страничного режима регистров цветов

0BH Различные возможности BIOS

Бит (ы)	Значение
0	= 1, если имеется световое перо
1	= 1, если возможно сохранение/восстановление видеосостояния (с платой MCGA всегда 0)
2	= 1, если возможно управление мерцанием/яркостью фона
3	= 1, если возможно получение/установка кода конфигурации видеосистемы
4–7	зарезервированы

0CH–0DH Зарезервированы

0EH Возможности области сохранения

Бит (ы)	Значение
0	= 1, если поддерживаются 512-символьные наборы
1	= 1, если имеется область динамического сохранения
2	= 1, если возможна замена алфавитного шрифта
3	= 1, если возможна замена графического шрифта
4	= 1, если возможна замена палитры
5	= 1, если имеется расширение кода конфигурации видеосистемы
6–7	зарезервированы

0FH Зарезервирован

Int 10H **[PS/2]**
Функция 1CH (28)
Сохранить или восстановить состояние видеосистемы

Сохраняет или восстанавливает состояние и регистры цветов цифро-аналогового преобразователя, область данных видеодрайвера ПЗУ BIOS или состояние аппаратуры видеосистемы.

При вызове. AH = 1CH
AL = подфункция
00H получить размер буфера состояния
01H сохранить состояние
02H восстановить состояние
CX = запрошенные состояния

Бит (ы)	**Значение** (если бит установлен)
0	сохранить/восстановить состояния аппаратуры видеосистемы
1	сохранить/восстановить области данных видео-BIOS
2	сохранить/восстановить состояние и регистры цветов цифро-аналогового видеопреобразователя
3–15	зарезервированы

ES : BX = сегмент : относительный_адрес буфера

При возврате. Если функция поддерживается
AL = 1CH
и, если при вызове AL = 00H
BX = счетчик блоков буфера (64 байт на блок)
или, если при вызове AL = 01H
Информация о состоянии, помещенная в буфер вызывающей задачи
или, если при вызове AL = 02H
Запрошенное состояние сохранено в соответствии с содержимым буфера вызывающей задачи

Замечания. Подфункция 00H используется для определения размера буфера, необходимого для сохранения указанной информации о состоянии. Буфер должен быть предоставлен вызывающей задачей.

Во время операции сохранения состояния (AL = 01H) текущее состояние видеосистемы изменяется. Если запрашивающей программе требуется продолжать выполнение при том же состоянии видеосистемы, непосредственно за запросом на сохранение состояния должен следовать запрос на восстановление состояния видеосистемы.

Эта функция поддерживается только видеоплатами VGA.

Int 11H **[PC] [AT] [PS/2]**
Получить конфигурацию оборудования

Позволяет получать из ПЗУ BIOS кодовое слово списка оборудования.

При вызове. Ничего не передается.

При возврате. AX = кодовое слово списка оборудования

Бит (ы)	**Значение**
0	= 1, если установлен гибкий диск (и)

Бит (ы)	Значение
1	= 1, если установлен арифметический сопроцессор
2	= 1, если установлено координатное устройство (PS/2)
2–3	размер ОЗУ системной платы (PC, см. *Замечание*) 00 = 16 Кбайт 01 = 32 Кбайт 10 = 48 Кбайт 11 = 64 Кбайт
4–5	число дисководов гибких дисков (если бит 0 = 1) 00 = 1 01 = 2 10 = 3 11 = 4
8	зарезервирован
9–11	число установленных портов RS-232
12	= 1, если установлен игровой адаптер
13	= 1, если установлен внутренний модем (только PC и XT) = 1, если подключен последовательный принтер (PCjr)
14–15	число установленных принтеров

Замечание. Биты 2–3 возвращаемого значения используются только в ПЗУ BIOS для исходных компьютеров IBM PC с памятью на системной плате 64 Кбайт, а также для PCjr.

Int 12H **[PC] [AT] [PS/2]**
Получить размер стандартной памяти

Возвращает объем стандартной памяти, доступной для использования MS-DOS и прикладными программами.
При вызове. Ничего не передается.
При возврате. AX = размер памяти (в килобайтах).
Замечания. В некоторых ранних моделях PC объем памяти, возвращаемый этой функцией, определяется микропереключателями на системной плате и может не соответствовать объему физически существующей памяти.

На машинах PC/AT возвращаемое значение равно объему функциональной памяти, обнаруженной во время самопроверки при включении питания, независимо от конфигурационной информации о размере памяти, содержащейся в КМОП–ОЗУ.

Возвращенное значение не отражает расширенной памяти (выходящей за границу 1 Мбайт), которая может быть установлена на машинах с микропроцессорами 80286 и 80386, таких, как PC/AT или PS/2 моделей 50 и выше.

Int 13H **[PC] [AT] [PS/2]**
Функция 00H
Привести в исходное состояние дисковую систему

Приводит в исходное состояние контроллер диска и подключенные к нему дисководы (позиционер головок чтения-записи перемещается к цилиндру 0) и подготавливает систему к операциям ввода-вывода на диске.
При вызове. AH = 00H
DL = дисковод

00H–7FH гибкий диск
80H–FFH жесткий диск

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечания. Функцию следует вызывать после сбоя при выполнении запросов на чтение, запись, верификацию или форматирование на гибком диске перед повторением операции.

Если при вызове функции DL = 80H (т.е. выбирается жесткий диск), приводятся в исходное состояние контроллеры гибкого и жесткого дисков. (См. также Int 13H с функцией 0DH, позволяющей привести в исходное состояние контроллер жесткого диска, не затрагивая гибкий диск.)

Int 13H **[PC] [AT] [PS/2]**
Функция 01H
Получить состояние дисковой системы

Возвращает состояние последней дисковой операции

При вызове. AH = 0H
DL = дисковод
00H–7FH гибкий диск
80H–FFH жесткий диск

При возврате. AH = 00H
AL = состояние предыдущей дисковой операции
00H отсутствие ошибки
01H недопустимая команда
02H не найдена адресная метка
03H диск защищен от записи (Г)
04H не найден сектор
05H сброс в исходное состояние не выполнен (Ж)
06H гибкий диск снят (Г)
07H дефектная таблица параметров (Ж)
08H нарушение границ ПДП (Г)
09H ПДП пересек границу 64 Кбайт
0AH флаг дефектного сектора (Ж)
0BH флаг дефектной дорожки (Ж)
0CH не найден тип носителя (Г)
0DH недопустимое число секторов при форматировании (Ж)
0FH обнаружена метка адреса управляющих данных (Ж)
уровень арбитража ПДП вышел из диапазона (Ж)
10H невосстановимая ошибка циклического контроля (ECC) или ошибка кода проверки и корректировки данных по избыточности (CRC)
11H ошибка данных скорректирована кодом проверки и корректировки (Ж)
20H отказ контроллера
40H сбой поиска
80H тайм-аут диска (отсутствие ответа)

AAH дисковод не готов (Ж)
BBH неопределенная ошибка (Ж)
CCH отказ записи (Ж)
EOH ошибка регистра состояния (Ж)
FFH операция опознания не выполнилась (Ж)
Г – только для гибкого диска; Ж – только для жесткого диска

Замечание. При использовании жесткого диска код ошибки 11H (ошибка скорректированных данных) указывает на обнаружение восстановимой ошибки в процессе предшествующего чтения сектора (Int 13H с функцией 02H).

Int 13H **[PC] [AT] [PS/2]**
Функция 02H
Прочитать сектор

Читает в память один или более секторов.
При вызове. AH = 02H
AL = число секторов
CH = цилиндр
CL = сектор
DH = головка
DL = дисковод
00H–7FH гибкий диск
80H–FFH жесткий диск
ES:BX = сегмент:относительный_адрес буфера
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
AL = число переданных секторов
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечания. При использовании жестких дисков старшие два бита 10-бит номера цилиндра помещаются в старшие два бита регистра CL.

При использовании жестких дисков код ошибки 11H свидетельствует о том, что ошибка чтения была скорректирована алгоритмом ECC (проверки и корректировки ошибки); в этом случае регистр AL содержит длину посылки. Возвращенные данные скорее всего правильны, хотя имеется незначительная вероятность неправильной корректировки. Если была запрошена многосекторная пересылка, операция завершилась передачей сектора, содержащего ошибку чтения.

При использовании гибких дисков ошибка может возникнуть, если к моменту запроса выключен мотор. ПЗУ BIOS не организует автоматического ожидания раскрутки мотора до требуемой скорости перед попыткой выполнения операции чтения. Запрашивающая программа должна установить в исходное состояние дисковую систему (Int 13H с функцией 00H) и трижды повторить операцию перед тем, как прийти к заключению, что ошибка вызвана чем-то другим.

Int 13H **[PC] [AT] [PS/2]**
Функция 03H
Записать сектор

Записывает из памяти на диск один или более секторов.

При вызове. AH = 03H
AL = число секторов
CH = цилиндр
CL = сектор
DH = головка
DL = дисковод
00H–7FH гибкий диск
80H–FFH жесткий диск

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
AL = число переданных секторов
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечания. При использовании жестких дисков старшие два бита 10-бит номера цилиндра помещаются в старшие два бита регистра CL.

При использовании гибких дисков ошибка может возникнуть, если к моменту запроса выключен мотор. ПЗУ BIOS не организует автоматического ожидания раскрутки мотора до требуемой скорости перед попыткой выполнения операции записи. Запрашивающая программа должна установить в исходное состояние дисковую систему (Int 13H с функцией 00H) и трижды повторить операцию перед тем, как прийти к заключению, что ошибка вызвана чем-то другим.

Int 13H **[PC] [AT] [PS/2]**
Функция 04H
Верифицировать сектор

Верифицирует адресные поля одного или более секторов. Эта операция не пересылает данные ни в память, ни из памяти.

При вызове. AH = 04H
AL = число секторов
CH = цилиндр
CL = сектор
DH = головка
DL = дисковод
00H–7FH гибкий диск
80H–FFH жесткий диск
ES:BX = сегмент:относительный_адрес буфера (см. *Замечания*)

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
AL = число верифицированных секторов
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечания. При использовании машин PC, PC/XT и PC/AT и ПЗУ BIOS, датированным ранее 15.11.85, ES:BX должен указывать на правильный буфер.

При использовании жестких дисков два старших бита 10-бит номера цилиндра помещаются в старшие два бита регистра CL.

Эту функцию можно использовать для проверки, установлен ли в дисководе гибких

дисков читаемый носитель. Ошибка может возникнуть, если к моменту запроса выключен мотор. ПЗУ BIOS не организует автоматического ожидания раскрутки мотора до требуемой скорости перед попыткой выполнения операции верификации. Запрашивающая программа должна установить в исходное состояние дисковую систему (Int 13H с функцией 00H) и трижды повторить операцию перед тем, как прийти к заключению, что читаемый гибкий диск отсутствует.

Int 13H **[PC] [AT] [PS/2]**
Функция 05H
Форматировать дорожку

Инициализирует поля адресов секторов и дорожки на указанной дорожке диска.
При вызове. AH = 05H
AL = чередование (жесткие диски PC/XT)
CH = цилиндр
DH = головка
DL = дисковод
00H–7FH гибкий диск
80H–FFH жесткий диск
ES:BX = сегмент : относительный_адрес списка адресных полей (кроме жестких дисков PC/XT см. *Замечание*)
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. На гибких дисках список адресных полей состоит из последовательности 4-бит записей, по одной записи на сектор, имеющих следующий формат:

Байт	Содержимое
0	Цилиндр
1	Головка
2	Сектор
3	Код размера сектора 00H, если 128 байт на сектор 01H, если 256 байт на сектор 02H, если 512 байт на сектор (стандарт) 03H, если 1024 байт на сектор

На гибких дисках число секторов на дорожку берется из таблицы параметров гибкого диска в BIOS, адрес которой записан в векторе прерывания Int 1EH.

Если эта функция используется для формирования гибких дисков на машинах PC/AT или PS/2, ей должен предшествовать вызов Int 13H с функцией 17H для выбора носителя, предназначенного для форматирования.

При использовании жестких дисков два старших бита 10-бит номера цилиндра помещаются в два старших бита регистра CL.

При использовании жестких дисков машин PC/XT-286, PC/AT и PS/2 регистр ES:BX указывает на 512-байт буфер, содержащий пары байтов для каждого сектора физического диска:

Байт	Содержимое
0	00H для работоспособного сектора 80H для дефектного сектора
1	номер сектора

Например, для форматирования дорожки с 17 секторами и чередованием, равным 2, ES:BX должен указывать на следующий 34-байт массив в начале 512-байт буфера:
db 00h,01h,00h,0ah,00h,02h,00h,0bh,00h,03h,00h,0ch
db 00h,04h,00h,0dh,00h,05h,00h,0eh,00h,06h,00h,0fh
db 00h,07h,00h,10h,00h,08h,00h,11h,00h,09h

Int 13H **[PC]**
Функция 06H
Форматировать дефектную дорожку

Инициализирует дорожку, записывая адресные поля диска и секторы данных и устанавливая флаги дефектных секторов.
При вызове. AH = 06H
AL = чередование
CH = цилиндр
DH = головка
DL = дисковод
80H – FFH жесткий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. Эта функция определена только для жестких дисков машин PC/XT.
См. *Замечания* к Int 13H с функцией 05H, где приведены дополнительные сведения.

Int 13H **[PC]**
Функция 07H
Форматировать дисковод

Форматирует весь дисковод, записывая адресные поля диска и секторы данных, начиная с указанного цилиндра.
При вызове. AH = 07H
AL = чередование
CH = цилиндр
DL = дисковод
80H – FFH жесткий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. Эта функция определена только для жестких дисков машин PC/XT.
См. *Замечания* к Int 13H с функцией 05H, где приведены дополнительные сведения.

Int 13H **[PC] [AT] [PS/2]**
Функция 08H
Получить параметры дисковода

Возвращает различные параметры указанного дисковода.
При вызове. AH = 08H
AL = дисковод
00H – 7FH гибкий диск
80H – FFH жесткий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
BL = тип дисковода (гибкие диски PC/AT и PS/2)
01H, если 360 Кбайт, 40 дорожек, 5,25″
02H, если 1,2 Мбайт, 80 дорожек, 5,25″
03H, если 720 Кбайт, 80 дорожек, 3,5″
04H, если 1,44 Мбайт, 80 дорожек, 3,5″
CH =младшие 8 бит максимального номера цилиндра
CL = биты 6 – 7: два старших бита максимального номера цилиндра
биты 0 – 5: максимальный номер сектора
DH =максимальный номер головки
DL =число дисководов
ES:DI = сегмент: относительный_адрес таблицы параметров дисковода
Если функция не выполнена
Флаг переноса установлен
AH =состояние (см. Int 13H с функцией 01H)

Замечания. На машинах PC и PC/XT эта функция поддерживается только на жестких дисках.

Значение, возвращаемое в регистре DL, отражает действительное число физических дисководов, прикрепленных к адаптеру запрошенного дисковода.

Int 13H **[PC] [AT] [PS/2]**
Функция 09H
Инициализировать характеристики жесткого диска

Инициализирует контроллер жесткого диска для последующих операций ввода-вывода с помощью значений, найденных в блоке (блоках) параметров диска ПЗУ.
При вызове. AH = 09H
DL = дисковод
80H – FFH жесткий диск
и на PC/XT
Вектор прерывания Int 41H должен указывать на блок параметров диска
или на PC/AT и PS/2
Вектор прерывания Int 41H должен указывать на блок параметров диска для дисковода 0
Вектор прерывания Int 46H должен указывать на блок параметров диска для дисковода 1
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена

Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечания. Эта функция поддерживается только на жестких дисках.

Для жестких дисков PC и PC/XT блок параметров имеет следующий формат:

Байт(ы)	Содержимое
00H–01H	Максимальное число цилиндров
02H	Максимальное число головок
03H–04H	Начальный текущий цилиндр укороченной записи
05H–06H	Начальный цилиндр предкомпенсации записи
07H	Максимальная длина посылки ECC
08H	Необязательные параметры дисковода

Бит(ы)	**Значение** (если установлено)
0–2	необязательные параметры дисковода
3–5	зарезервированы (0)
6	запретить повторные попытки ECC
7	запретить повторные попытки обращения к диску

09H	стандартное значение тайм-аута
0AH	значение тайм-аута для форматирования диска
0BH	значение тайм-аута для проверки диска
0CH–0FH	зарезервированы

Для жестких дисков PC/AT и PS/2 блок параметров имеет следующий формат:

Байт(ы)	Содержимое
00H–01H	Максимальное число цилиндров
02H	Максимальное число головок
03H–04H	Зарезервированы
05H–06H	Начальный цилиндр предкомпенсации записи
07H	Максимальная длина посылки ECC
08H	Необязательные параметры дисковода

Бит(ы)	**Значение** (если установлено)
0–2	не используются
3	более 8 головок
4	не используется
5	дефектная карта изготовителя расположена по адресу максимальный цилиндр + 1
6–7	не нули (10, 01 или 11), если запрещены повторные попытки

09H–0BH	Зарезервированы
0CH–0DH	Цилиндр зоны парковки
0EH	Секторов на дорожку
0FH	Зарезервированы

Int 13H **[PC] [AT] PS/2]**
Функция 0AH(10)
Прочитать расширенный сектор

Читает с диска в память сектор или секторы вместе с 4-байт кодом ECC для каждого сектора.

При вызове. AH = 0AH
AL = число секторов
CH = цилиндр
CL = сектор (см. *Замечания*)
DH = головка
DL = дисковод
80H – FFH жесткий диск
ES:BX = сегмент: относительный_адрес буфера
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
AL = число переданных секторов
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. Эта функция поддерживается только на жестких дисках.

Два старших бита 10-бит номера цилиндра помещаются в два старших бита регистра CL. В отличие от функции нормального чтения сектора (Int 13H с функцией 02H) ошибки ECC не корректируются автоматически. Многосекторная передача завершается после передачи любого сектора с ошибкой чтения.

Int 13H **[PC] [AT] [PS/2]**
Функция 0BH(11)
Записать расширенный сектор

Записывает из памяти на диск сектор или секторы. За каждым сектором данных должен следовать 4-байт код ECC.
При вызове. AH = 0BH
AL = число секторов
CH = цилиндр
CL = сектор (см. *Замечания*)
DH = головка
DL = дисковод
80H – FFH жесткий диск
ES:BX = сегмент: относительный_диск буфера
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
AL = число переданных секторов
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. Эта функция поддерживается только на жестких дисках.

Два старших бита 10-битового номера цилиндра размещаются в двух старших битах регистра CL.

Int 13H **[PC] [AT] [PS/2]**
Функция 0CH(12)
Установка головок

Устанавливает головки чтения-записи на указанный цилиндр, но не передает данных.

При вызове. AH = 0CH
CH = младшие 8 бит номера цилиндра
CL = старшие 2 бит номера цилиндра в битах 6–7
DH = головка
DL = дисковод
80H – FFH жесткий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. Эта функция поддерживается только на жестких дисках.

Два старших бита 10-бит номера цилиндра размещаются в двух старших битах регистра CL.

Функции «Прочитать сектор», «Прочитать расширенный сектор», «Записать сектор» и «Записать расширенный сектор» включают в себя операцию поиска и не требуют перед своим выполнением явного обращения к этой функции.

Int 13H **[PC] [AT] [PS/2]**
Функция 0DH (13)
Привести в исходное состояние систему жесткого диска

Приводит в исходное состояние контроллер жесткого диска и подключенные к нему дисководы (позиционер головок чтения-записи перемещается к цилиндру 0), а также подготавливает систему к операциям ввода-вывода на диск.
При вызове. AH = 0DH
DL = дисковод
80H – FFH жесткий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. Эта функция поддерживается только на жестких дисках. От Int 13H с функцией 00H она отличается тем, что контроллер гибкого диска не приводится в исходное состояние.

Int 13H **[PC]**
Функция 0EH(14)
Прочитать буфер сектора

Пересылает содержимое внутреннего буфера сектора адаптера жесткого диска в системную память. Не происходит считывания данных с физического диска.
При вызове. AH = 0EH
ES:BX = сегмент: относительный_адрес буфера
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен

AH = состояние (см. Int 13H с функцией 01H)

Замечание. Эта функция поддерживается только адаптером жесткого диска машин PC/XT. Она не определена для адаптеров жестких дисков машин PC/AT или PS/2.

Int 13H **[PC]**
Функция 0FH (15)
Записать буфер сектора

Пересылает данные из системной памяти во внутренний буфер сектора адаптера жесткого диска. Не происходит записи данных на физический диск.

При вызове. AH = 0FH
ES:BX = сегмент: относительный_адрес буфера

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечания. Эта функция поддерживается только адаптером жестких дисков машин PC/XT. Для адаптеров жестких дисков машин PC/AT или PS/2 она не определена.

Эту функцию следует вызывать для инициализации содержимого буфера сектора перед форматированием диска с помощью прерывания Int 13H с функцией 05H.

Int 13H **[PC] [AT] [PS/2]**
Функция 10H(16)
Получить состояние дисковода

Проверяет, является ли указанный дисковод жесткого диска работоспособным и возвращает его состояние.

При вызове. AH = 10H
DL = дисковод
80H–FFH жесткий диск

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечание. Эта функция поддерживается только на жестких дисках.

Int 13H **[PC] [AT] [PS/2]**
Функция 11H(17)
Привести в исходное положение позиционер дисковода

Заставляет адаптер жесткого диска привести в исходное состояние позиционер головок чтения-записи указанного дисковода, переместив его к цилиндру 0, и возвращает состояние дисковода.

При вызове. AH = 11H
DL = дисковод
80H–FFH жесткий диск

При возврате. Если функция выполнена успешно
Флаг переноса сброшен

AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечание. Эта функция поддерживается только на жестких дисках.

Int 13H **[PC]**
Функция 12H(18)
Диагностика ОЗУ контроллера

Заставляет адаптер жесткого диска выполнить встроенный диагностический тест содержимого его внутреннего буфера сектора и сообщить о результатах теста возвращаемым состоянием.

При вызове. AH = 12H

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечание. Эта функция поддерживается только на жестких дисках машин PC/XT.

Int 13H **[PC]**
Функция 13H(19)
Диагностика контроллера дисковода

Заставляет адаптер жесткого диска выполнить внутренний диагностический тест подключенного дисковода и сообщить о его результатах возвращаемым состоянием.

При вызове. AH = 13H

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечание. Эта функция поддерживается только на жестких дисках машин PC/XT.

Int 13H **[PC] [AT] [PS/2]**
Функция 14H(20)
Внутренняя диагностика контроллера

Заставляет адаптер жесткого диска выполнить встроенную процедуру диагностического самотестирования и сообщить о ее результатах возвращаемым состоянием.

При вызове. AH = 14H

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)

Замечание. Эта функция поддерживается только на жестких дисках.

Int 13H **[PC] [PS/2]**
Функция 15H(21)
Получить тип диска

Возвращает код указанного дисковода, соответствующий типу гибкого или жесткого диска.

При вызове. AH = 15H
DL = дисковод
00H – 7FH гибкий диск
80H – FFH жесткий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = код типа дисковода
00H, если дисковод отсутствует
01H, если используется дисковод гибкого диска без поддержки сигнала смены носителя
02H, если используется дисковод гибкого диска с поддержкой сигнала смены носителя
03H, если используется жесткий диск
и, если используется жесткий диск (AH = 03H)
CX:DX = число 512-байт секторов
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечание. Эта функция не поддерживается на машинах PC или PC/XT.

Int 13H **[AT] [PS/2]**
Функция 16H(22)
Получить состояние смены диска

Возвращает состояние сигнала смены носителя, указывающего, могла ли произойти смена диска с момента последнего обращения к нему.
При вызове. AH = 16H
DL = дисковод
00H – 7FH гибкий диск
При возврате. Если сигнал смены неактивен (смены диска не было)
Флаг переноса сброшен
AH = 00H
Если сигнал смены активен (возможно, произошла смена диска)
Флаг переноса установлен
AH = 06H
Замечания. Если после выполнения этой функции флаг переноса установлен, не обязательно произошла смена диска; сигнал смены мог быть активизирован просто открыванием и закрыванием дверцы дисковода без смены дискеты.

Эта функция не поддерживается на гибких дисках на машинах PC или PC/XT.

Int 13H **[AT] [PS/2]**
Функция 17H(23)
Установить тип диска

Выбирает тип гибкого диска для указанного дисковода.

При вызове. AH = 17H
AL = код типа гибкого диска
00H не используется
01H дискета емкостью 320/360 Кбайт на дисководе на 360 Кбайт
02H дискета емкостью 320/360 Кбайт на дисководе на 1,2 Мбайт
03H дискета емкостью 1,2 Мбайт на дисководе на 1,2 Мбайт
04H дискета емкостью 720 Кбайт на дисководе на 720 Кбайт
DL = дисковод
00H–7FH гибкий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. Эта функция не поддерживается на гибких дисках на машинах PC или PC/XT.

Если для указанного дисковода сигнал смены активен, дисковод приводится в исходное состояние. Затем ПЗУ BIOS устанавливает скорость передачи данных для указанного дисковода и типа носителя.

Int 13H **[AT] [PS/2]**
Функция 18H(24)
Установить тип носителя для форматирования

Выбирает характеристики носителя для указанного дисковода.
При вызове. AH = 18H
CH = число цилиндров
CL = число секторов на дорожке
DL = дисковод
00H–7FH гибкий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
ES:DI = сегмент:относительный_адрес таблицы параметров диска для типа носителя
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. На дисководе должен быть установлен гибкий диск.

Эту функцию следует вызывать перед форматированием диска с помощью прерывания Int 13H с функцией 05H, чтобы программы ПЗУ BIOS могли установить для конкретного носителя правильное значение скорости передачи данных.

Если для указанного дисковода сигнал смены активен, дисковод приводится в исходное состояние.

Int 13H **[PS/2]**
Функция 19H(25)
Парковать головки

Перемещает позиционер головок чтения-записи на дорожку, не используемую для хранения данных, чтобы исключить повреждение данных при выключении дисковода.

При вызове. AH = 19H
DL = дисковод
80H–FFH жесткий диск
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 0IH)
Замечание. Эта функция определена только для жестких дисков машин PS/2.

Int 13H **[PS/2]**
Функция 1AH(26)
Форматировать дисковод ESDI

Инициализирует адресные поля секторов и дорожек диска на дисководе, подключенном к плате Adapter/A ESDI-дисковода жестких дисков.
При вызове. AH = 1AH
AL = счетчик таблицы дефектов относительных адресов блоков (RBA)
0, если таблица RBA отсутствует
> 0, если используется таблица RBA
CL = биты модификатора формата

Бит(ы)	**Значение** (если бит установлен)
0	игнорировать первичную карту дефектов
1	игнорировать вторичную карту дефектов
2	обновить вторичную карту дефектов (см. *Замечания*)
3	выполнить расширенный анализ поверхности
4	генерировать периодическое прерывание (см. *Замечания*)
5–7	зарезервированы (должны содержать 0)

DL = дисковод
80H–FFH жесткий диск
ES:BX = сегмент:относительный_адрес таблицы
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 13H с функцией 01H)
Замечания. Эту операцию иногда называют форматированием низкого уровня; она подготавливает диск на уровне секторов к операциям физического чтения-записи. Далее диск должен быть разбит на разделы с помощью команды FDISK и, наконец, подвергнут «форматированию высокого уровня» командой FORMAT с целью установки файловой системы. Если установлен бит 4 регистра CL, после форматирования или анализа каждого цилиндра вызывается прерывание Int 15H при AH = 0FH и AL = код фазы. Последний определяется следующим образом:

0 = зарезервирован
1 = анализ поверхности
2 = форматирование

См. также Int 15H с функцией 0FH.

Если установлен бит 2 регистра CL, вторичная карта дефектов диска обновляется с целью отражения ошибок, найденных во время анализа поверхности. Если установлены оба бита, 2 и 1, вторичная карта дефектов заменяется.

С целью выполнения расширенного анализа поверхности диск следует сначала отформатировать вызовом этой функции со сброшенным битом 3, а затем проанализировать вызовом этой же функции с установленным битом 3.

Int 14H **[PC] [AT] [PS/2]**
Функция 00H
Инициализировать коммуникационный порт

Инициализирует последовательный коммуникационный порт, устанавливая требуемые скорость передачи, четности, длину слова и число стоп-битов.

При вызове. AH = 00H
AL = параметр инициализации (см. *Замечания*)
DX = номер коммуникационного порта (0 = COM1, 1 = COM2 и т. д.)

При возврате. AH = состояние порта

Бит	**Значение** (если бит установлен)
0	готовность приемника данных
1	обнаружена ошибка наложения
2	обнаружена ошибка контроля четности
3	обнаружена ошибка кадра
4	обнаружен разрыв (break)
5	регистр данных передатчика пуст
6	сдвиговый регистр передатчика пуст
7	тайм-аут

AL = состояние модема

Бит	**Значение** (если бит установлен)
0	изменение состояния сигнала «готов к передаче»
1	изменение состояния сигнала «аппаратура передачи данных готова»
2	индикатор завершения вызова
3	изменение состояния детектора принимаемого сигнала
4	готов к передаче
5	аппаратура передачи данных готова
6	индикатор вызова
7	детектор принимаемого сигнала

Замечания. Байт параметра инициализации определяется следующим образом:

765	**43**	**2**	**10**
Скорость передачи	**Четность**	**Стоп-биты**	**Длина**
000 = 110	X0 = отсутствует	0 = 1 бит	10 = 7 бит
001 = 150	01 = проверка на нечетность	1 = 2 бит	11 = 8 бит
010 = 300	11 = проверка на четность		
011 = 600			
100 = 1200			
101 = 2400			
110 = 4800			
111 = 9600			

Инициализация последовательного порта на машинах PS/2 для передачи данных со скоростями, превышающими 9600 бод, осуществляется с помощью прерывания Int 14H с функциями 04H и 05H.

Int 14H **[PC] [AT] [PS/2]**
Функция 01H
Записать символ в коммуникационный порт

Записывает символ в указанный последовательный коммуникационный порт, возвращая текущее состояние порта.
При вызове. AH = 01H
AL = символ
DX = номер коммуникационного порта (0 = COM1, 1 = COM2 и т. д.)
При возврате. Если функция выполнена успешно
AH, бит 7 = 0
AH, биты 0–6 = состояние порта

Бит	Значение (если бит установлен)
0	готовность приемника данных
1	обнаружена ошибка наложения
2	обнаружена ошибка четности
3	обнаружена ошибка кадра
4	обнаружен разрыв (break)
5	регистр данных передатчика пуст
6	сдвиговый регистр передатчика пуст

AL = символ (не изменяется)
Если функция не выполнена (тайм-аут)
AH, бит 7 = 1
AL = символ (не изменяется)

Int 14H **[PC] [AT] [PS/2]**
Функция 02H
Прочитать символ из коммуникационного порта

Читает символ из указанного последовательного коммуникационного порта и возвращает состояние порта.
При вызове. AH = 02H
DX = номер коммуникационного порта (0 = COM1, 1 = COM2 и т. д.)
При возврате. Если функция выполнена успешно
AH, бит 7 = 0
AH, биты 0–6 = состояние

Бит	Значение (если бит установлен)
1	обнаружена ошибка наложения
2	обнаружена ошибка четности
3	обнаружена ошибка кадра
4	обнаружен разрыв (break)

AL = символ
Если функция не выполнена (тайм-аут)
AH, бит 7 = 1

Int 14H **[PC] [AT] [PS/2]**
Функция 03H
Получить состояние коммуникационного порта

Возвращает состояние указанного последовательного коммуникационного порта.
При вызове. AH = 03H
DX = номер коммуникационного порта (0 = COM1, 1 = COM2 и т. д.)
При возврате. AH = состояние порта (см. Int 14H с функцией 00H)
AL = состояние модема (см. Int 14H с функцией 00H)

Int 14H **[PS/2]**
Функция 04H
Расширенная инициализация коммуникационного порта

Инициализирует последовательный коммуникационный порт, устанавливая требуемые значения скорости передачи, четности, длины слова и числа стоп-битов. Реализует подмножество возможностей прерывания Int 14H с функцией 00H для машин PS/2.
При вызове. AH = 04H
AL = флаг разрыва (break)
00H разрыв отсутствует
01H разрыв
BH = четность
00H отсутствует
01H проверка на нечетность
02H проверка на четность
03H стик-паритет нечетен
04H стик-паритет четен
BL = стоп-биты
00H 1 стоп-бит
01H 2 стоп-бита, если длина слова = 6–8 бит
02H 1,5 стоп-бита, если длина слова = 5 бит
CH = длина слова
00H 5 бит
01H 6 бит
02H 7 бит
03H 8 бит
CL = скорость передачи
00H 110 бод
01H 150 бод
02H 300 бод
03H 600 бод
04H 1200 бод
05H 2400 бод
06H 4800 бод
07H 9600 бод
08H 19200 бод
DX = номер коммуникационного порта (0 = COM1, 1 = COM2 и т. д.)
При возврате. AH = состояние порта (см. Int 14H с функцией 00H)
AL = состояние модема (см. Int 14H с функцией 00H)

Int 14H **[PS/2]**
Функция 05H
Расширенное управление коммуникационным портом

Читает или устанавливает регистр управления модемом для указанного последовательного коммуникационного порта.
При вызове. AH = 05H
AL = подфункция
00H чтение регистра управления модемом
01H запись в регистр управления модемом
BL = содержимое регистра управления модемом (если AL = 01H)

Бит(ы)	Значение
0	оконечное оборудование данных готово
1	запрос передачи
2	выход 1
3	выход 2
4	контрольный режим для проверки (вход подключен к выходу)
5–7	зарезервированы

DX = номер коммуникационного порта (0 = COM1, 1 = COM2 и т. д.)
При возврате. Если при вызове AL = 00H
BL = содержимое регистра управления модемом (см. выше)
Если при вызове AL = 01H
AH = состояние порта (см. Int 14H с функцией 00H)
AL = состояние модема (см. Int 14H с функцией 00H)

Int 15H **[PC]**
Функция 00H
Включить мотор кассеты

Включает мотор запоминающего устройства на магнитной ленте кассетного типа.
При вызове. AH = 00H
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен
AH = состояние
86H, если кассета отсутствует
Замечание. Эта функция может использоваться только на машинах PC и PCjr. Она не поддерживается на машинах PC/XT и всех последующих моделях.

Int 15H **[PC]**
Функция 01H
Выключить мотор кассеты

Выключает мотор запоминающего устройства на магнитной ленте кассетного типа.
При вызове. AH = 01H
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен

AH = состояние
86H, если кассета отсутствует

Замечание. Эта функция может использоваться только на машинах PC и PCjr. Она не поддерживается на машинах PC/XT и всех последующих моделях.

Int 15H **[PC]**
Функция 02H
Прочитать кассету

Читает один или более 256-байт блоков данных из запоминающего устройства на магнитной ленте кассетного типа в память.

При вызове. AH = 02H
CX = число читаемых байтов
ES:BX = сегмент:относительный_адрес буфера

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
DX = число реально прочитанных байтов
ES:BX = сегмент:относительный_адрес + 1 последнего прочитанного байта
Если функция не выполнена
Флаг переноса установлен
AH = состояние
01H, если ошибка CRC
02H, если битовые сигналы закодированы
04H, если данные не найдены
80H, если неправильная команда
86H, если кассета отсутствует

Int 15H **[PC]**
Функция 03H
Записать кассету

Записывает один или более 256-байт блоков данных из памяти на магнитную ленту кассетного типа.

При вызове. AH = 03H
CX = число записываемых байтов
ES:BX = сегмент:относительный_адрес буфера

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
CX = 00H
ES:BX = сегмент:относительный адрес + 1 последнего записанного блока
Если функция не выполнена
Флаг переноса установлен
AH = состояние
80H, если недопустимая команда
86H, если кассета отсутствует

Замечание. Эта функция может использоваться только на машинах PC и PCjr. Она не поддерживается на машинах PC/XT и всех последующих моделях.

Int 15H [PS/2]
Функция 0FH(15)
Периодическое прерывание при форматировании ESDI-дисковода

Возбуждается программами ПЗУ BIOS при использовании платы Adapter/A ESDI-дисковода жестких дисков в процессе форматирования или анализа поверхности после завершения обработки каждого цилиндра.
При вызове. AH = 0FH
AL = код фазы
0 = зарезервирован
1 = анализ поверхности
2 = форматирование
При возврате. Если форматирование или анализ следует продолжить
Флаг переноса сброшен
Если форматирование или анализ следует завершить
Флаг переноса установлен
Замечания. Этот функциональный вызов может быть «отловлен» программой с целью оповещения об окончании форматирования или анализа каждого цилиндра. Программа может считать прерывания каждой фазы для определения номера текущего цилиндра.

Обработчик этого прерывания в ПЗУ BIOS, используемый по умолчанию, возвращает установленный флаг переноса.

Int 15H [PS/2]
Функция 21H(33), подфункция 00H
Прочитать журнал ошибок POST

Возвращает информацию об ошибках, накопленную во время последнего самотестирования при включении питания (POST).
При вызове. AH = 21H
AL = 00H
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
BX = число записанных кодов ошибок POST
ES:DI = сегмент:относительный_адрес журнала ошибок POST
Если функция не выполнена
Флаг переноса установлен
AH = состояние
80H = неправильная команда
86H = функция не поддерживается
Замечания. Журнал ошибок состоит из однословных записей. Первый байт каждой записи содержит код ошибки устройства, второй – идентификатор устройства.

Эта функция отсутствует на машинах PS/2 моделей 25 и 30.

Int 15H [PS/2]
Функция 21H(33), подфункция 01H
Записать журнал ошибок POST

Добавляет запись в журнал ошибок самотестирования при включении питания (POST).
При вызове. AH = 21H

AL = 01H
BH = идентификатор устройства
BL = код ошибки устройства

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние
01H = список ошибок полон
80H = недопустимая команда
86H = функция не поддерживается

Замечание. Эта функция отсутствует на машинах PS/2 моделей 25 и 30.

Int 15H **[PS/2]**
Функция 4FH(79)
Перехват клавиатуры

Вызывается при каждом нажатии клавиши обработчиком прерываний от клавиатуры Int 09H, расположенным в ПЗУ BIOS.

При вызове. AH = 4FH
AL = скен-код

При возврате. Если скен-код поглощен
Флаг переноса сброшен
Если скен-код не поглощен
AL = неизменный или новый скен-код

Замечания. Операционная система или резидентная утилита может перехватывать эту функцию с целью фильтрации исходного потока данных от клавиатуры. Новый обработчик может подставить новый скен-код, вернуть тот же скен-код или вернуть сброшенный флаг переноса, что вызывает отбрасывание конкретного нажатия клавиши. Программу ПЗУ BIOS, действующая по умолчанию, просто возвращает скен-код в неизменном виде.

Программа может вызвать Int 15H с функцией C0H для того, чтобы определить, поддерживает ли ПЗУ BIOS в используемой машине этот перехват клавиатуры.

Int 15H **[AT] [PS/2]**
Функция 80H(128)
Устройство открыто

Приобретает для процесса права использования логического устройства.

При вызове. AH = 80H
BX = идентификатор (ID) устройства
CX = идентификатор (ID) процесса

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние

Замечания. Этот функциональный вызов вместе с прерыванием Int 15H с функциями 81H и 82H определяет простой протокол, с помощью которого можно выполнять

арбитраж использования устройств многими процессами. Администратор многозадачного режима должен перехватывать прерывания Int 15H и обеспечивать необходимое обслуживание. Программа BIOS, используемая по умолчанию, при выполнении этой функции просто возвращает сброшенный флаг переноса и AH = 00H.

Int 15H **[AT] [PS/2]**
Функция 81H(129)
Устройство закрыто

Освобождает права использования процессом логического устройства.
При вызове. AH = 81H
BX = идентификатор (ID) устройства
CX = идентификатор (ID) процесса
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние
Замечания. Администратор многозадачного режима должен перехватывать прерывания Int 15H и обеспечивать необходимое обслуживание. Программа BIOS, используемая по умолчанию, при выполнении этой функции просто возвращает сброшенный флаг переноса и AH = 00H. (См. также Int 15H с функциями 80H и 82H.)

Int 15H **[AT] [PS/2]**
Функция 82H(130)
Завершение процесса

Освобождает права использования любых логических устройств завершающимся процессом.
При вызове. AH = 82H
BX = идентификатор (ID) процесса
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние
Замечания. Администратор многозадачного режима должен перехватывать прерывания Int 15H и обеспечивать соответствующее обслуживание. Программа BIOS этой функции, используемая по умолчанию, просто возвращает сброшенный флаг переноса и AH = 00H. (См. также Int 15H с функциями 80H и 81H.)

Int 15H **[AT] [PS/2]**
Функция 83H(131)
Ожидание события

Запрашивает установку семафора после истечения заданного интервала либо отменяет предыдущий запрос.
При вызове. Если запрос на ожидание события
AH = 83H

AL = 00H
CX:DX = микросекунды
ES:BX = сегмент:относительный_адрес байта семафора
Если отмена ожидания события
AH = 83H
AL = 01H

При возврате. Если при вызове AL = 00H и функция выполнена успешно
Флаг переноса сброшен
Если при вызове AL = 00H и функция не выполнена (ожидание события уже активно)
Флаг переноса установлен
Если при вызове AL = 01H
Ничего не возвращается

Замечания. Возврат из функции осуществляется немедленно. Если функция выполнилась успешно, бит 7 байта семафора устанавливается после истечения заданного интервала. Сброс семафора перед вызовом этой функции лежит на вызывающей программе.

Фактическая продолжительность ожидания события всегда является целым числом, кратным 976 мкс. Для реализации этой функции используются прерывания от КМОП-микросхемы даты/времени.

Использование этой функции позволяет получить программируемые аппаратно-независимые задержки с более высоким временным разрешением, чем с помощью функции MS-DOS «Получить время» (Int 21H с функцией 2CH), возвращающей время в единицах сотых долей секунды.

См. также Int 15H с функцией 86H, которая задерживает выполнение программы на заданный (в миллисекундах) интервал времени.

Эта функция не поддерживается на машинах PS/2 моделей 25 и 30.

Int 15H **[AT] [PS/2]**
Функция 84H(132)
Прочитать джойстик

Возвращает состояние переключателей манипулятора «джойстик» и отсчеты, снятые с его потенциометров.

При вызове. AH = 84H
DX = подфункция
00H – прочитать состояние переключателей
01H – прочитать отсчеты потенциометров

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
и, если при вызове DX = 00H
AL = состояние переключателей (биты 4–7)
или, если при вызове DX = 01H
AX = значение A(x)
BX = значение A(y)
CX = значение B(x)
DX = значение B(y)
Если функция не выполнена
Флаг переноса установлен

Замечания. Если DX содержит недопустимый номер подфункции, возвращается условие ошибки.

Если игровой адаптер не установлен, для подфункции 00H в AL возвращается 00H, (т.е. все переключатели разомкнуты); для подфункции 01H в AX, BX, CX, и DX возвращается 00H.

При использовании 250-кОм джойстика отсчеты потенциометров обычно лежат в диапазоне 0–416 (0000–01A0H).

Int 15H **[AT] [PS/2]**
Функция 85H(133)
Клавиша SysReq

Активизируется драйвером клавиатуры ПЗУ BIOS, когда обнаруживается нажатие или отпускание клавиши SysReq.

При вызове. AH = 85H
AL = состояние клавиши
00H – клавиша нажимается
01H – клавиша отпускается

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние

Замечание. Обработчик ПЗУ BIOS этого функционального вызова представляет собой пустую программу, всегда возвращающую состояние успешного выполнения, если только она не была вызвана с недопустимым кодом подфункции в регистре AL. Администратор многозадачного режима должен перехватывать прерывание Int 15H, чтобы он мог узнать о нажатии пользователем клавиши SysReq.

Int 15H **[AT] [PS/2]**
Функция 86H(134)
Задержка

Приостанавливает выполнение вызывающей программы на заданный (в микросекундах) интервал времени.

При вызове. AH = 86H
CX:DX = интервал ожидания в микросекундах

При возврате. Если функция выполнена успешно (приостановка состоялась)
Флаг переноса сброшен
Если функция не выполнена (приостановка не состоялась)
Флаг переноса установлен

Замечания. Фактическая продолжительность интервала ожидания всегда является целым числом, кратным 976 мкс.

Использование этой функции позволяет получить программируемые аппаратно-независимые задержки с более высоким временным разрешением, чем с помощью функции MS-DOS «Получить время» (Int 21H с функцией 2CH), возвращающей время в единицах сотых долей секунды.

См. также Int 15H с функцией 83H, которая после истечения заданного интервала времени устанавливает семафор, но не приостанавливает выполнение программы.

Int 15H **[AT] [PS/2]**
Функция 87H(135)
Переслать блок расширенной памяти

Пересылает данные между обычной и расширенной памятью.
При вызове. AH = 87H
CX = число пересылаемых слов
ES:SI = сегмент:относительный_адрес таблицы глобальных дескрипторов (см. *Замечания*)
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние
01H – ошибка четности ОЗУ
02H – ошибка прерывания по исключительному состоянию
03H – отказ линии 20 адреса вентиля

Замечания. Обычная память расположена по адресам, не превышающим граничного значения 640 Кбайт; она используется для выполнения программ MS-DOS и прикладных программ. Расширенная память расположена по адресам, большим 1 Мбайт; доступ к ней реализован только в ЦП на базе микропроцессоров 80286 и 80386, работающих в защищенном режиме. На машинах IBM PC/AT и их аналогах можно установить до 15 Мбайт расширенной памяти.

Таблица глобальных дескрипторов (GDT), используемая этой функцией, должна содержать следующую информацию:

Байт(ы)	Содержимое
00H–0FH	Зарезервированы (должны содержать 0)
10H–11H	Длина сегмента в байтах (2*CX–1 или больше)
12H–14H	24-разрядный адрес источника
15H	Байт прав доступа (всегда 93H)
16H–17H	Зарезервированы (должны содержать 0)
18H–19H	Длина сегмента в байтах (2*CX–1 или больше)
1AH–1CH	24-разрядный адрес приемника
1DH	Байт прав доступа (всегда 93H)
1EH–2FH	Зарезервированы (должны содержать 0)

Таблица состоит из шести 8-байт дескрипторов, используемых ЦП в защищенном режиме. Четыре дескриптора со смещениями 00H–0FH и 20H–2FH заполняются программами ПЗУ BIOS перед переключением режима ЦП.

Адреса, используемые в таблице дескрипторов, являются линейными (физическими) 24-разрядными адресами в диапазоне 000000H–FFFFFFH (не сегменты и относительные адреса). Младший байт располагается по наименьшему адресу, старший – по наибольшему. При пересылке блока прерывания запрещаются; таким образом, использование этой функции может помешать выполнению коммуникационных программ, драйверов сети и другого программного обеспечения, требующего быстрого обслуживания аппаратных прерываний.

Программы и драйверы, обращающиеся с помощью этой функции к расширенной памяти, не могут выполняться в среде DOS-совместимости системы OS/2.

Эта функция не поддерживается на машинах PS/2 моделей 25 и 30.

Int 15H **[AT] [PS/2]**
Функция 88H(136)
Получить размер расширенной памяти

Возвращает объем расширенной памяти, установленной в системе.
При вызове. AH = 88H
При возврате. AX = объем расширенной памяти (в килобайтах)
Замечания. Расширенной называется память, расположенная по адресам, большим 1 Мбайт; доступ к ней реализован только в ЦП на базе микропроцессоров 80286 и 80386, работающих в защищенном режиме. Поскольку MS-DOS – это операционная система реального режима, расширенная память может использоваться для временного хранения данных, но не для выполнения программ.

Программы и драйверы, обращающиеся с помощью этой функции к расширенной памяти, не могут выполняться в среде DOS-совместимости OS/2.

Эта функция не поддерживается на машинах PS/2 моделей 25 и 30.

Int 15H **[AT] [PS/2]**
Функция 89H(137)
Вход в защищенный режим

Переключает ЦП из реального режима в защищенный.
При вызове. AH = 89H
BH = номер прерывания для IRQ0, записанный в ICW2 программируемого контроллера прерываний 8259 номер 1 (должен нацело делиться на 8, определяет IRQ0 – IRQ7)
BL = номер прерывания для IRQ8, записанный в ICW2 программируемого контроллера прерываний 8259 номер 2 (должен нацело делиться на 8, определяет IRQ8 – IRQ15)
ES:SI = сегмент:относительный_адрес таблицы глобальных дескрипторов (GDT)
При возврате. Если функция выполнена успешно (ЦП находится в защищенном режиме)
Флаг переноса сброшен
AH = 00H
CS = селектор, определяемый пользователем
DS = селектор, определяемый пользователем
ES = селектор, определяемый пользователем
SS = селектор, определяемый пользователем
Если функция не выполнена (ЦП находится в реальном режиме)
Флаг переноса установлен
AH = FFH
Замечания. Таблица глобальных дескрипторов должна содержать следующие восемь дескрипторов:

Относительный адрес	**Использование дескриптора**
00H	Пустой дискриптор (инициализирован нулем)
08H	Таблица глобальных дескрипторов (GDT)
10H	Таблица дескрипторов прерываний (IDT)
18H	Сегмент данных пользователя (DS)
20H	Экстра-сегмент пользователя (ES)

Относительный адрес	Использование дескриптора
28H	Сегмент стека пользователя (SS)
30H	Программный сегмент пользователя (CS)
38H	Программный сегмент BIOS

Пользователь должен инициализировать первые семь дескрипторов; восьмой заполняется программой ПЗУ BIOS для обеспечения адресуемости при выполнении самих программ BIOS. Вызывающая программа может модифицировать и использовать восьмой дескриптор в любых целях после возврата из этого функционального вызова.

Эта функция не поддерживается на машинах PS/2 моделей 25 и 30.

Int 15H **[AT] [PS/2]**
Функция 90H(144)
Ожидание устройства

Активизируется драйверами ПЗУ BIOS обслуживания жесткого диска, гибкого диска, принтера, сети и клавиатуры перед выполнением программного ожидания завершения ввода-вывода.

При вызове. AH = 90H
AL = тип устройства
00H – 7FH последовательно повторно-используемые устройства
80H – BFH повторно-входимые устройства
C0H – FFH запросы только на ожидание, соответствующая функция окончания отсутствует
ES:BX = сегмент:относительный_адрес блока запроса для устройств типов 80H – FFH

При возврате. Если запрос на ожидание не поставлен (драйвер должен сам организовать ожидание)
Флаг переноса сброшен
AH = 00H
Если запрос на ожидание поставлен
Флаг переноса установлен

Замечания. Предопределенные типы устройств таковы:

00H	Диск (тайм-аут реализуется)
01H	Гибкий диск (тайм-аут реализуется)
02H	Клавиатура (тайм-аут не реализуется)
03H	Координатное устройство (PS/2, тайм-аут реализуется)
80H	Сеть (тайм-аут не реализуется)
FCH	Установка в исходное состояние жесткого диска (PS/2, тайм-аут реализуется)
FDH	Запуск мотора дисковода гибких дисков (тайм-аут реализуется)
FEH	Принтер (тайм-аут не реализуется)

При использовании сетевых адаптеров ES:BX указывает на сетевой управляющий блок (NCB).

Администратор многозадачного режима должен перехватывать прерывание Int 15H с целью обеспечения диспетчеризации других задач в процессе выполнения ввода-вывода. Программа BIOS реализации этой функции, используемая по умолчанию, просто возвращает сброшенный флаг переноса и AH = 00H.

Int 15H **[AT] [PS/2]**
Функция 91H(145)
Устройство закончило

Активизируется драйверами ПЗУ BIOS обслуживания жесткого диска, гибкого диска, сети и клавиатуры для оповещения о завершении ввода-вывода и/или готовности устройства.
При вызове. AH = 91H
AL = тип устройства
00H – 7FH последовательно повторно-используемые устройства
80H – BFH повторно-входимые устройства
ES:BX = сегмент:относительный_адрес блока запроса для устройств типов 80H – BFH
При возврате. AH = 00H
Замечания. Предопределенные типы устройств таковы:

00H	Диск (тайм-аут реализуется)
01H	Гибкий диск (тайм-аут реализуется)
02H	Клавиатура (тайм-аут не реализуется)
03H	Координатное устройство (PS/2, тайм-аут реализуется)
80H	Сеть (тайм-аут не реализуется)

Программа ПЗУ BIOS обслуживания принтера не активизирует эту функцию, потому что вывод принтера не управляется прерываниями.

Администратор многозадачного режима должен перехватывать прерывание Int 15H с функцией 91H, чтобы он мог узнать о завершении ввода-вывода и разблокировать запрашивающую задачу. Программа BIOS реализации этой функции, используемая по умолчанию, просто возвращает сброшенный флаг переноса и AH = 00H.

Int 15H **[AT] [PS/2]**
Функция C0H(192)
Получить системную среду

Возвращает указатель таблицы, содержащей разнообразную информацию о конфигурации системы.
При вызове. AH = C0H
При возврате. ES:BX = сегмент:относительный_адрес таблицы (см. *Замечания*)
Замечания. Таблица конфигурации системы имеет следующий формат:

Байт(ы)	**Содержимое**
00H – 01H	Длина таблицы в байтах
02H	Модель системы (см. следующее *Замечание*)
03H	Подмодель системы (см. следующее *Замечание*)
04H	Уровень ревизии BIOS
05H	Флаги конфигурации

Бит	**Значение** (если установлен)
0	зарезервирован
1	реализована шина Micro Channel
2	выделена расширенная область даных BIOS
3	реализована функция «Ожидание внешнего события»

Бит	Значение (если установлен)
4	реализована функция перехвата клавиатуры (Int 15H с функцией 4FH)
5	имеются часы реального времени
6	присутствует ведомый контроллер 8259 (каскадированное прерывание IRQ2)
7	используется канал прямого доступа к памяти номер 3

06H–09H Зарезервированы

Байты модели и типа системы заполняются следующим образом:

Машина	Байт модели	Байт подмодели
PC	FFH	
PC/XT	FEH	
PC/XT	FBH	00H или 01H
PCjr	FDH	
PC/AT	FCH	00H или 01H
PC/XT–286	FCH	02H
PC Convertible	F9H	
PS/2, модель 30	FAH	00H
PS/2, модель 50	FCH	04H
PS/2, модель 60	FCH	05H
PS/2, модель 70	F8H	04H или 09H
PS/2, модель 80	F8H	00H или 01H

Int 15H **[PS/2]**
Функция C1H(193)
Получить адрес расширенной области данных BIOS

Возвращает адрес сегмента базы расширенной области данных BIOS.
При вызове. AH = 01H
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
ES = сегмент расширенной области данных
Если функция не выполнена
Флаг переноса установлен
Замечания. Расширенная область данных BIOS назначается в конце обычной памяти в процессе выполнения последовательности самотестирования при включении питания (POST). Слово по адресу 0040:0013H (размер памяти) обновляется, чтобы отразить уменьшение объема памяти, доступной MS-DOS и прикладным программам. Первый байт расширенной области данных BIOS инициализируется длиной области в килобайтах. Программа может получить сведения о наличии расширенной области данных BIOS с помощью прерывания Int 15H с функцией C0H.

Int 15H **[PS/2]**
Функция C2H(194), подфункция 00H
Разблокировать/заблокировать координатное устройство

Разблокирует или блокирует системное устройство мышь или другое координатное устройство.

При вызове. AH = C2H
AL = 00H
BH = флаг разблокирования/блокирования
00H = заблокировать
01H = разблокировать
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние
01H, если недопустимый функциональный вызов
02H, если недопустимый ввод
03H, если ошибка интерфейса
04H, если повторная передача
05H, если дальний вызов не установлен

Int 15H **[PS/2]**
Функция C2H, подфункция 01H
Привести в исходное состояние координатное устройство

Приводит в исходное состояние системное устройство мышь или другое координатное устройство, инициализируя частоту выборки, разрешение и другие характеристики значениями, используемыми по умолчанию.
При вызове. AH = C2H
AL = 01H
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
BH = идентификатор устройства (ID)
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 15H с функцией C2H и подфункцией 00H)
Замечания. После выполнения этой функции состояние координатного устройства характеризуется следующим образом:
– заблокировано;
– частота выборки 100 посылок в секунду;
– разрешение 4 отсчета на миллиметр;
– масштабирование 1 : 1.
Размер пакета данных этой функцией не изменяется.

Прикладная программа может использовать прерывание Int 15H с функцией C2H и другими подфункциями для инициализации координатного устройства другими значениями частоты выборки, разрешения и масштабирования, а затем разблокировать устройство с помощью прерывания Int 15H с функцией C2H и подфункцией 00H.

См. также Int 15H с функцией C2H и подфункцией 05H, приводящее координатное устройство в исходное состояние аналогичным образом.

Int 15H **[PS/2]**
Функция C2H(194), подфункция 02H
Установить частоту выборки

Устанавливает частоту выборки системного устройства мышь или другого координатного устройства.

При вызове. AH = C2H
AL = 02H
BH = частота выборки
00H = 10 посылка/с
01H = 20 посылка/с
02H = 40 посылка/с
03H = 60 посылка/с
04H = 80 посылка/с
05H = 100 посылка/с
06H = 200 посылка/с

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 15H с функцией C2H и подфункцией 00H)

Замечание. Частота выборки по умолчанию после операции приведения в исходное состояние (Int 15H с функцией C2H и подфункцией 01H) составляет 100 посылок в секунду.

Int 15H **[PS/2]**
Функция C2H(194), подфункция 03H
Установить разрешение

Устанавливает разрешение системного устройства мышь или другого координатного устройства.

При вызове. AH = C2H
AL = 03H
BH = значение разрешения
01H = 1 отсчет/мм
02H = 2 отсчет/мм
03H = 4 отсчет/мм
04H = 8 отсчет/мм

При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 15H с функцией C2H и подфункцией 00H)

Замечание. Разрешение по умолчанию после операции приведения в исходное состояние (Int 15H с функцией C2H и подфункцией 01H) составляет 4 отсчета на миллиметр.

Int 15H **[PS/2]**
Функция C2H(194), подфункция 04H
Получить тип координатного устройства

Возвращает код идентификации системного устройства мышь или другого координатного устройства.
При вызове. AH = C2H
Al = 04H
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
BH = код идентификации (ID) устройства
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 15H с функцией C2H и подфункцией 00H)

Int 15H **[PS/2]**
Функция C2H(194), подфункция 05H
Инициализировать интерфейс координатного устройства

Устанавливает для системного устройства мышь или другого координатного устройства размер пакета данных, а также инициализирует разрешение, частоту выборки и масштабирование значениями, используемыми по умолчанию.
При вызове. AH = C2H
AL = 05H
BH = размер пакета данных в байтах (1–8)
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 15H с функцией C2H и подфункцией 00H)
Замечание. После этой операции состояние координатного устройства характеризуется следующим образом:
– заблокировано;
– частота выборки 100 посылок в секунду;
– разрешение 4 отсчета на миллиметр;
– масштабирование 1 : 1.

Int 15H **[PS/2]**
Функция C2H(194), подфункция 06H
Установить масштаб или получить состояние

Возвращает текущее состояние системного устройства мышь или другого координатного устройства либо устанавливает коэффициент масштабирования устройства.
При вызове. AH = C2H
AL = 06H
BH = расширенная команда
00H = вернуть состояние устройства
01H = установить масштабирование 1 : 1
02H = установить масштабирование 2 : 1

При возврате. Если функция выполнилась успешно
Флаг переноса сброшен
AH = 00H
и, если при вызове BH = 00H
BL = байт состояния

Бит	Значение
0	= 1, если нажата правая клавиша
1	зарезервирован
2	= 1, если нажата левая клавиша
3	зарезервирован
4	= 0, если масштабирование 1 : 1
	1, если масштабирование 2 : 1
5	= 0, если устройство заблокировано
	1, если устройство разблокировано
6	= 0, если стримерный режим
	1, если удаленный режим
7	зарезервирован

CL = разрешение
00H = 1 отсчет/мм
01H = 2 отсчет/мм
02H = 4 отсчет/мм
03H = 8 отсчет/мм
DL = частота выборки
0AH = 10 посылка/с
14H = 20 посылка/с
28H = 40 посылка/с
3CH = 60 посылка/с
50H = 80 посылка/с
64H = 100 посылка/с
C8H = 200 посылка/с
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 15H с функцией C2H и подфункцией 00H)

Int 15H **[PS/2]**
Функция C2H(194), подфункция 07H
Установить адрес обработчика прерываний от координатного устройства

Передает драйверу координатного устройства (из ПЗУ BIOS) адрес программы, которую нужно вызывать при каждом получении данных от координатного устройства.
При вызове. AH = C2H
AL = 07H
ES : BX = сегмент : относительный_адрес программы пользователя
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен
AH = состояние (см. Int 15H с функцией C2H и подфункцией 00H)
Замечания. Обработчик пользователя для работы с данными от координатного устройства вызывается командой дальнего вызова с четырьмя параметрами на стеке:

SS:SP + 0AH состояние
SS:SP + 08H x-координата
SS:SP + 06H y-координата
SS:SP + 04H z-координата (всегда 0)

Выход из обработчика осуществляется командой дальнего возврата без снятия параметров со стека.

Параметры состояния, передаваемые в обработчик пользователя, интерпретируются следующим образом:

Бит(ы)	**Значение** (если бит установлен)
0	нажата левая клавиша
1	нажата правая клавиша
2–3	зарезервированы
4	знак x-координаты отрицателен
5	знак y-координаты отрицателен
6	переполнение x-координаты
7	переполнение y-координаты
8–15	зарезервированы

Int 15H **[PS/2]**
Функция C3H(195)
Установить сторожевой тайм-аут

Разблокирует или блокирует сторожевой таймер.
При вызове. AH = C3H
AL = подфункция
00H заблокировать сторожевой тайм-аут
01H разблокировать сторожевой тайм-аут
BX = счетчик сторожевого таймера (если AL = 01H)
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен
Замечания. Сторожевой таймер генерирует немаскируемые прерывания (NMI).
Эта функция отсутствует на машинах PS/2 моделей 25 и 30.

Int 15H **[PS/2]**
Функция C4H(196)
Выбор программируемых опций

Возвращает адрес базового регистра выбора программируемых опций, разблокирует разъем для установки или адаптер.
При вызове. AH = C4H
AL = подфункция
00H – вернуть адрес базового регистра выбора программируемых опций
01H – разблокировать разъем
02H = разблокировать адаптер
BL = номер разъема (если AL = 01H)
При возврате. Если функция выполнена успешно
Флаг переноса сброшен

и, если при вызове AL = 00H
DX = адрес базового регистра адаптера выбора программируемых опций
Если функция не выполнена
Флаг переноса установлен

Замечания. Эта функция может использоваться только на машинах с шиной Micro Channel Architecture (MCA).

После разблокировки разъема с помощью подфункции 01H можно, выполнив операции ввода из портов, получить специфическую информацию об адаптере:

Порт	Функция
1000H	идентификатор MCA (младший байт)
101H	идентификатор MCA (старший байт)
102H	байт 1 выбора опций
	бит 0 = 1, если разблокирован
	= 0, если заблокирован
103H	байт 2 выбора опций
104H	байт 3 выбора опций
105H	байт 4 выбора опций
	биты 6–7 = индикаторы проверки канала
106H	расширение субадреса (младший байт)
107H	расширение субадреса (старший байт)

Int 16H **[PC] [AT] [PS/2]**
Функция 00H
Прочитать символ с клавиатуры

Читает символ с клавиатуры, возвращая также скен-код клавиши.
При вызове. AH = 00H
При возврате. AH = скен-код клавиши
AL = символ ASCII

Int 16H **[PC] [AT] [PS/2]**
Функция 01H
Получить состояние клавиатуры

Определяет, имеется ли готовый к вводу символ, возвращает флаг, а также и сам символ, если символ ожидал ввода.
При вызове. AH = 01H
При возврате. Если имеется ожидающий ввода код клавиши
Флаг нуля сброшен
AH = скен-код клавиши
AL = символ
Если нет ожидающего кода
Флаг нуля установлен

Замечание. Символ, возвращаемый этой функцией при сброшенном флаге нуля, не извлекается из буфера клавиатуры. Те же самые символ и скен-код клавиши будут возвращены следующим вызовом Int 16H с функцией 00H.

Int 16H **[PC] [AT] [PS/2]**
Функция 02H
Получить флаги клавиатуры

Возвращает байт флагов ПЗУ BIOS, описывающий состояние различных переключателей и клавиш клавиатуры.
При вызове. AH = 02H
При возврате. AL = флаги

Бит	**Значение** (если установлено)
0	правая клавиша Shift нажата
1	левая клавиша Shift нажата
2	клавиша Ctrl нажата
3	клавиша Alt нажата
4	переключатель Scroll Lock включен
5	переключатель Num Lock включен
6	переключатель Caps Lock включен
7	переключатель Insert включен

Замечание. Байт флагов клавиатуры хранится в области данных ПЗУ BIOS по адресу 0000:0417H.

Int 16H **[PC] [AT] [PS/2]**
Функция 03H
Установить скорость повторения

Устанавливает скорость повторения символа нажатой клавиши и задержку повторения («автопечать»), определяемые ПЗУ BIOS.
При вызове. На машинах PC/AT и PS/2
AH = 03H
AL = 05H
BH = задержка повторения (см. *Замечания*)
BL = скорость повторения (см. *Замечания*)
На машинах PCjr
AH = 03H
AL = подфункция
00H = восстановить значения скорости и задержки повторения, действующие по умолчанию
01H – увеличить начальную задержку
02H – уменьшить скорость повторения вдвое
03H – увеличить задержку и уменьшить скорость повторения вдвое
04H – выключить повторение нажатой клавиши
При возврате. Ничего не возвращает.
Замечания. Подфункции 00H – 04H используются на машинах PCjr, но не поддерживаются ПЗУ BIOS машин PC или PC/XT. Подфункцию 05H можно использовать на машинах PC/AT с ПЗУ BIOS, датированным 15.11.85 и позже, а также на машинах PS/2.

На машинах PC/AT и PS/2 значение в регистре BH определяет задержку до генерации первого повторного кода. Задержка всегда кратна 250 мс:

Значение	**Задержка, мс**
00H	250

Значение	Задержка, мс
01H	500
02H	750
03H	1000

На машинах PC/AT и PS/2 значение скорости повторения в единицах символ/с может быть выбрано из следующей таблицы:

Значение	Скорость повторения (символ/с)
00H	30,0
01H	26,7
02H	24,0
03H	21,8
04H	20,0
05H	18,5
06H	17,1
07H	16,0
08H	15,0
09H	13,3
0AH	12,0
0BH	10,9
0CH	10,0
0DH	9,2
0EH	8,6
0FH	8,0
10H	7,5
11H	6,7
12H	6,0
13H	5,5
14H	5,0
15H	4,6
16H	4,3
17H	4,0
18H	3,7
19H	3,3
1AH	3,0
1BH	2,7
1CH	2,5
1DH	2,3
1EH	2,1
1FH	2,0

Int 16H **[PC]**
Функция 04H
Установить звуковой сигнал клавиш

Включает и выключает звуковой сигнал клавиш.
При вызове. AH = 04H
AL = подфункция
00H – включить звуковой сигнал клавиш
01H – выключить звуковой сигнал клавиш

При возврате. Ничего не возвращается.
Замечание. Эта функция поддерживается только BIOS машин PCjr.

Int 16H **[AT] [PS/2]**
Функция 05H
Протолкнуть символ и скен-код

Помещает символ и скен-код клавиши в буфер клавиатуры.
При вызове. AH = 05H
CH = скен-код клавиши
CL = символ
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
AH = 00H
Если функция не выполнена (буфер клавиатуры заполнен)
Флаг переноса установлен
AL = 01H
Замечание. Эта функция может использоваться расширителями клавиатуры и другими утилитами для включения кодов клавиш в поток данных, воспринимаемых прикладной программой.

Int 16H **[AT] [PS/2]**
Функция 10H(16)
Прочитать символ с расширенной клавиатуры

Считывает символ и скен-код клавиши из буфера клавиатуры.
При вызове. AH = 10H
При возврате. AH = скен-код клавиши
AL = символ ASCII
Замечание. Используйте эту функцию вместо Int 16H с функцией 00H при работе с расширенной клавиатурой. Функция дает возможность получать скен-коды дополнительных клавиш F11 и F12, а также клавиш управления курсором.

Int 16H **[AT] [PS/2]**
Функция 11H(17)
Получить состояние расширенной аппаратуры

Определяет, имеет ли готовый к вводу символ, возвращает флаг, а также и сам символ, если символ ожидал ввода.
При вызове. AH = 11H
При возврате. Если имеется ожидающий ввода код клавиши
Флаг нуля сброшен
AH = скен-код клавиши
AL = символ
Если нет ожидающего кода
Флаг нуля установлен
Замечания. Используйте эту функцию вместо Int 16H с функцией 00H при работе с расширенной клавиатурой. Функция дает возможность получать скен-коды дополнительных клавиш F11 и F12, а также клавиш управления курсором.

Символ, получаемый этой функцией при сброшенном флаге нуля, не извлекается из буфера клавиатуры. Те же самые символ и скен-код клавиши будут возвращены следующим вызовом Int 16H с функцией 00H.

Int 16H **[AT] [PS/2]**
Функция 12H(18)
Получить флаги расширенной клавиатуры

Позволяет получить состояния различных специальных клавиш расширенной клавиатуры и драйвера клавиатуры.
При вызове. AH = 12H
При возврате. AX = флаги

Бит	**Значение** (если бит установлен)
0	правая клавиша Shift нажата
1	левая клавиша Shift нажата
2	любая из клавиш Ctrl нажата
3	любая из клавиш Alt нажата
4	переключатель Scroll Losk включен
5	переключатель Num Lock включен
6	переключатель Caps Lock включен
7	переключатель Insert включен
8	левая клавиша Ctrl нажата
9	левая клавиша Alt нажата
10	правая клавиша Ctrl нажата
11	правая клавиша Alt нажата
12	клавиша Scroll нажата
13	клавиша Num Lock нажата
14	клавиша Caps Lock нажата
15	клавиша SysReq нажата

Замечание. Используйте эту функцию для работы с расширенной клавиатурой вместо Int 16H с функцией 02H.

Int 17H **[PC] [AT] [PS/2]**
Функция 00H
Записать символ на принтер

Пересылает символ в указанный порт параллельного интерфейса принтера и возвращает текущее состояние порта.
При вызове. AH = 00H.
AL = символ
DX = номер принтера (0 = LPT1, 1 = LPT2, 2 = LPT3)
При возврате. AH = состояние

Бит	**Значение** (если бит установлен)
0	в принтере возникла ситуация тайм-аута
1	не используется
2	не используется
3	ошибка ввода-вывода
4	принтер выбран
5	нет бумаги
6	подтверждение принтера
7	принтер не занят

Int 17H **[PC] [AT] [PS/2]**
Функция 01H
Инициализировать порт принтера

Инициализирует указанный порт параллельного интерфейса принтера и возвращает его состояние.
При вызове. AH = 01H
DX = номер принтера (0 = LPT1, 1 = LPT2, 2 = LPT3)
При возврате. AH = состояние (см. Int 17H с функцией 00H)

Int 17H **[PC] [AT] [PS/2]**
Функция 02H
Получить состояние принтера

Возвращает текущее состояние указанного порта параллельного интерфейса принтера.
При вызове. AH = 02H
DX = номер принтера (0 = LPT1, 1 = LPT2, 2 = LPT3)
При возврате. AH = состояние (см. Int 17H с функцией 00H)

Int 18H **[PC] [AT] [PS/2]**
ПЗУ системы Бейсик
Передает управление ПЗУ системы Бейсик

При вызове. Ничего не передается.
При возврате. Ничего не возвращается.
Замечание. Эта функция запускается при включении или перезагрузке системы, если не удалось прочитать сектор загрузчика с жесткого или гибкого диска.

Int 19H **[PC] [AT] [PS/2]**
Перезагрузка системы
Перезагружает операционную систему с гибкого или жесткого диска

При вызове. Ничего не передается.
При возврате. Ничего не возвращается.
Замечания. Программа начальной загрузки считывает сектор 1 дорожки 0 в память по адресу 0000:7C00H и передает управление по этому адресу. Если попытки прочитать сектор загрузчика с гибкого или жесткого диска окончились безуспешно, управление передается ПЗУ системы Бейсик возбуждением прерывания Int 18H.

Если ячейка с адресом 0000:0472H не содержит значение 1234H, перед чтением сектора загрузчика выполняется тест памяти.

Int 1AH **[AT] [PS/2]**
Функция 00H
Получить число тактов

Возвращает содержимое счетчика тактов таймера.
При вызове. AH = 00H
При возврате. AL = флаг перехода через полночь
00H – с момента последнего чтения не было перехода через полночь
≠ 00H – с момента последнего чтения был переход через полночь

CX:DX = число тактов (старше 16 бит в CX)

Замечания. Эта функция поддерживается BIOS машин PC/XT и PCjr, но не включена в ПЗУ BIOS исходных машин PC.

Возвращаемое значение представляет собой накопленное число тактов таймера после полуночи. Их частота составляет 18,2 такт/с. Когда содержимое счетчика достигает 1 573 040, он очищается, и устанавливается флаг перехода через полночь.

Этот функциональный вызов приводит к сбросу флага перехода через полночь, который, таким образом, может быть возвращен установленным только один раз в сутки.

Для занесения в счетчик тактов таймера произвольного 32-разрядного значения можно воспользоваться вызовом Int 1AH с функцией 01H.

Int 1AH **[AT] [PS/2]**
Функция 01H
Установить число тактов

Записывает 32-разрядное число в счетчик тактов таймера.

При вызове. AH = 01H
CX:DX = число тактов (старше 16 бит в CX)

При возврате. Ничего не возвращается.

Замечания. Эта функция поддерживается BIOS машин PC/XT и PCjr, но не включена в ПЗУ BIOS исходных машин PC.

Для чтения содержимого счетчика тактов таймера используется вызов Int 1AH с функцией 00H.

Эта функция сбрасывает флаг перехода через полночь.

Int 1AH **[AT] [PS/2]**
Функция 02H
Получить время

Считывает значение текущего времени с КМОП-микросхемы отсчета времени/даты.

При вызове. AH = 02H

При возврате. CH = часы в двоично-десятичном коде
CL = минуты в двоично-десятичном коде
DH = секунды в двоично-десятичном коде
DL = код летнего времени
00H – стандартное время
01H – летнее время
и, если таймер работает
Флаг переноса сброшен
или, если таймер остановлен
Флаг переноса установлен

Int 1AH **[AT] [PS/2]**
Функция 03H
Установить время

Устанавливает значение времени в КМОП-микросхеме отсчета времени/даты.

При вызове. AH = 03H
CH = часы в двоично-десятичном коде
CL = минуты в двоично-десятичном коде

DH = секунды в двоично-десятичном коде
DL = код летнего времени
00H – стандартное время
01H – летнее время
При возврате. Ничего не возвращается.

Int 1AH **[AT] [PS/2]**
Функция 04H
Получить дату

Считывает значение текущего времени с КМОП-микросхемы отсчета времени/даты.
При вызове. AH = 04H
CH = столетие (19 или 20) в двоично-десятичном коде
CL = год в двоично-десятичном коде
DH = месяц в двоично-десятичном коде
DL = день в двоично-десятичном коде
и, если таймер работает
Флаг переноса сброшен
или, если таймер остановлен
Флаг переноса установлен

Int 1AH **[AT] [PS/2]**
Функция 05H
Установить дату

Устанавливает дату в КМОП-микросхеме отсчета времени/даты.
При вызове. AH = 05H
CH = столетие (19 или 20) в двоично-десятичном коде
CL = год в двоично-десятичном коде
DH = месяц в двоично-десятичном коде
DL = день в двоично-десятичном коде
При возврате. Ничего не возвращается.

Int 1AH **[AT] [PS/2]**
Функция 06H
Установить будильник

Устанавливает на определенное время будильник в КМОП-микросхеме отсчета времени/даты.
При вызове. AH = 06H
CH = часы в двоично-десятичном коде
CL = минуты в двоично-десятичном коде
DH = секунды в двоично-десятичном коде
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена (будильник уже установлен или таймер остановлен)
Флаг переноса установлен
Замечания. В качестве побочного эффекта при вызове этой функции разблокируется уровень прерываний микросхемы таймера (IRQ8).

В каждый данный момент может быть активным только один запрос срабатывания

будильника. Сигнал будильника возникает каждые 24 часа в указанное время, пока будильник не будет сброшен с помощью Int 1AH с функцией 07H.

Программа, использующая эту функцию, должна поместить адрес своего обработчика прерывания от будильника в вектор прерывания Int 4AH.

Int 1AH **[AT] [PS/2]**
Функция 07H
Сбросить будильник

Отменяет установленный ранее запрос к будильнику в КМОП-микросхеме отсчета времени/даты.
При вызове. AH = 07H
При возврате. Ничего не возвращается.
Замечание. Эта функция не блокирует уровень прерываний микросхемы таймера (IRQ8).

Int 1AH **[PS/2]**
Функция 0AH(10)
Получить содержимое счетчика дней

Возвращает содержимое системного счетчика дней.
При вызове. AH = 0AH
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
CX = число дней, отсчитанных от 1 января 1980 г.
Если функция не выполнена
Флаг переноса установлен

Int 1AH **[PS/2]**
Функция 0BH(11)
Установить счетчик дней

Записывает произвольное значение в системный счетчик дней.
При вызове. AH = 0BH
CX = число дней от 1 января 1980 г.
При возврате. Если функция выполнена успешно
Флаг переноса сброшен
Если функция не выполнена
Флаг переноса установлен

Int 1AH **[PC]**
Функция 80H(128)
Установить источник звука

Устанавливает источник музыкальных тонов, поступающих на выход „Audio Out" или радиочастотный модулятор машины PCjr.
При вызове. AH = 80H
AL = источник звука
00H – программируемый таймер 8253, канал 2
01H – кассетный вход

02H – линия „Audio In" канала ввода-вывода
03H – микросхема звукового генератора

При возврате. Ничего не возвращается.

Замечание. Эта функция поддерживается только на машинах PCjr.

Int 33H
Драйвер мыши фирмы Microsoft

Функции драйвера мыши фирмы Microsoft доступны прикладным программам через прерывание Int 33H. Эти функции стали стандартом де факто для координатных устройств всех видов. В отличие от остальных функциональных вызовов, описанных в этом разделе, драйвер мыши фирмы Microsoft не входит в состав ПЗУ BIOS, а загружается с помощью директивы DEVICE-файла CONFIG.SYS. Вся информация о функциях обслуживания мыши относится к версии 6 драйвера мыши фирмы Microsoft. Ранние версии драйвера могут не поддерживать некоторые из этих функций.

Int 33H
Функция 00H
Привести мышь в исходное состояние и получить состояние драйвера

Инициализирует драйвер мыши и возвращает его состояние. Если указатель мыши был виден на экране, он удаляется с экрана, и все предварительно установленные пользовательские обработчики событий мыши блокируются.

При вызове. AX = 0000H

При возврате. Если имеется поддержка мыши
AX = FFFFH
BX = число клавиш мыши
Если не имеется поддержки мыши
AX = 0000H

Замечание. После этого функционального вызова драйвер мыши инициализируется следующим образом:

– указатель мыши находится в центре экрана (см. Int 33H с функциями 03H и 04H);

– для отображения указателя мыши назначается видеостраница 0 (см. Int 33H с функциями 1DH и 1EH);

– указатель мыши удаляется с экрана (см. Int 33H с функциями 01H, 02H и 10H);

– указателю мыши назначается действующая по умолчанию форма стрелки в графических режимах или негативного прямоугольника в текстовых режимах (см. Int 33H с функциями 09H и 0AH);

– пользовательский обработчик событий мыши блокируется (см. Int 33H с функциями 0CH и 14H);

– эмуляция светового пера разблокируется (см. Int 33H с функциями 0DH и 0EH);

– коэффициент преобразования по горизонтали устанавливается равным 8 : 8 микки на пиксел, по вертикали – 16 : 8 микки на пиксел (см. Int 33H с функцией 0FH);

– порог удвоения скорости устанавливается равным 64 микки/с (см. Int 33H с функцией 1AH);

– минимальное и максимальное значения горизонтального и вертикального положения курсора устанавливаются так, чтобы в текущем видеорежиме включить весь экран (см. Int 33H с функциями 07H и 08H).

Int 33H
Функция 01H
Вывести на экран указатель мыши

Выводит на экран указатель мыши и отменяет любую область подавления указателя, установленную ранее с помощью Int 33H с функцией 10H.
При вызове. AX = 0001H
При возврате. Ничего не возвращается.
Замечание. Система поддерживает счетчик, над которым вызов Int 33H с функцией 02H (погасить указатель мыши) выполняет дескремент, а настоящая функция – инкремент (если в счетчике не 0). Если в счетчике 0 или становится 0, указатель мыши изображается на экране. При установке в исходное состояние драйвера мыши с помощью прерывания Int 33H с функцией 00H в счетчик заносится –1.

Int 33H
Функция 02H
Погасить указатель мыши

Гасит изображение указателя мыши на экране. Драйвер продолжает отрабатывать положение мыши.
При вызове. AX = 0002H
При возврате. Ничего не возвращается.
Замечание. Система поддерживает счетчик, над которым настоящая функция выполняет декремент, а вызов Int 33H с функцией 01H («Вывести на экран указатель мыши») – инкремент (если в счетчике не 0). Если в счетчике 0 или становится 0, указатель мыши изображается на экране. При сбросе драйвера мыши с помощью прерывания Int 33H с функцией 00H в счетчик заносится –1.

Int 33H
Функция 03H
Получить положение мыши и состояние клавиш

Возвращает текущее состояние клавиш мыши и положение указателя.
При вызове. AX = 0003H
При возврате. BX = состояние клавиш мыши

Бит(ы)	**Значение** (если бит установлен)
0	левая клавиша нажата
1	правая клавиша нажата
2	центральная клавиша нажата
3–15	зарезервированы (0)

CX = горизонтальная (X) координата
DX = вертикальная (Y) координата
Замечание. Значения координат возвращаются в пикселах независимо от текущего видеорежима. Положение (x, y) = (0, 0) соответствует верхнему левому углу экрана.

Int 33H
Функция 04H
Установить положение указателя мыши

Устанавливает положение указателя мыши. Изображение указателя появляется на экране в указанном месте, если только он не погашен с помощью прерывания Int 33H

с функцией 02H или не попал в область подавления, определенную с помощью прерывания Int 33H с функцией 10H.

При вызове. AX = 0004H
CX = горизонтальная (X) координата
DX = вертикальная (Y) координата

При возврате. Ничего не возвращается.

Замечания. Значения координат указываются в пикселах независимо от текущего видеорежима. Положение (x, y) = (0, 0) соответствует верхнему левому углу экрана.

При необходимости положение корректируется так, чтобы не выходить за горизонтальные и вертикальные границы, установленные предшествующим вызовом Int 33H с функциями 07H и 08H.

Int 33H
Функция 05H
Получить информацию о нажатиях клавиш мыши

Возвращает текущее состояние всех клавиш мыши, а также число нажатий и координаты последнего нажатия заданной клавиши мыши с момента последнего вызова этой функции применительно к заданной клавише. Счетчик нажатий конкретной клавиши сбрасывается в 0.

При вызове. AX = 0005H
BX = идентификатор клавиши
0 = левая клавиша
1 = правая клавиша
2 = средняя клавиша

При возврате. AX = состояние клавиши

Бит(ы)	**Значение** (если бит установлен)
0	левая клавиша нажата
1	правая клавиша нажата
2	средняя клавиша нажата
3–15	зарезервированы

BX = счетчик нажатий клавиши
CX = горизонтальная (X) координата последнего нажатия клавиши
DX = вертикальная (Y) координата последнего нажатия клавиши

Int 33H
Функция 06H
Получить информацию об отпусканиях клавиш мыши

Возвращает текущее состояние всех клавиш мыши, а также число отпусканий и координаты последнего отпускания заданной клавиши мыши с момента последнего вызова этой функции применительно к заданной клавише. Счетчик отпусканий конкретной клавиши сбрасывается в 0.

При вызове. AX = 0006H
BX = идентификатор клавиши
0 = левая клавиша
1 = правая клавиша
2 = средняя клавиша

При возврате. AX = состояние клавиши

Бит(ы)	Значение (если бит установлен)
0	левая клавиша нажата
1	правая клавиша нажата
2	средняя клавиша нажата
3–15	зарезервированы (0)

BX = счетчик отпускания клавиши
CX = горизонтальная (X) координата последнего отпускания клавиши
DX = вертикальная (Y) координата последнего отпускания клавиши

Int 33H
Функция 07H
Установить горизонтальные границы для указателя

Ограничивает область изображения на экране указателя мыши назначением минимальной и максимальной горизонтальных (X) координат указателя.
При вызове. AX = 0007H
CX = минимальная горизонтальная (X) координата
DX = максимальная горизонтальная (X) координата
При возврате. Ничего не возвращается.
Замечания. Если минимальное значение больше максимального значения, оба значения взаимно меняются местами.

Положение указателя мыши будет при необходимости изменено так, чтобы оно лежало внутри указанных горизонтальных координат.

См. также Int 33H с функцией 10H, которая определяет область подавления указателя мыши.

Int 33H
Функция 08H
Установить вертикальные границы для указателя

Ограничивает область изображения на экране указателя мыши назначением минимальной и максимальной вертикальных (Y) координат указателя.
При вызове. AX = 0008H
CX = минимальная вертикальная (Y) координата
DX = максимальная вертикальная (Y) координата
При возврате. Ничего не возвращается.
Замечания. Если минимальное значение больше максимального, оба значения взаимно меняются местами.

Положение указателя мыши будет при необходимости изменено так, чтобы оно лежало внутри указанных вертикальных координат.

См. также Int 33H с функцией 10H, которая определяет область подавления указателя мыши.

Int 33H
Функция 09H
Установить форму графического указателя

Определяет форму, цвет и «горячую точку» указателя мыши в графических режимах.
При вызове. AX = 0009H
BX = смещение «горячей точки» от левого края
CX = смещение «горячей точки» от верхнего края

ES:DX = сегмент:относительный_адрес буфера образа указателя

При возврате. Ничего не возвращается.

Замечания. Буфер образа указателя имеет длину 64 байт. Первые 32 байт содержат битовую маску, выполняющую с изображением на экране операцию логического И; вторые 32 байт содержат битовую маску, выполняющую с изображением на экране логическую операцию ИСКЛЮЧАЮЩЕЕ ИЛИ.

«Горячая точка» определяется относительно верхнего левого угла образа указателя, причем каждое из смещений (в пикселах) должно лежать в пределах от −16 до 16. В видеорежимах 4 и 5 горизонтальное смещение должно быть четным.

Int 33H
Функция 0AH(10)
Установить тип текстового указателя

Определяет форму и атрибуты указателя мыши в текстовых режимах.

При вызове. AX = 000AH
BX = тип указателя
0 = программный курсор
1 = аппаратный курсор
CX = значение маски И (если BX = 0) или начальная строка курсора (если BX = 1)
DX = значение маски ИЛИ (если BX = 0) или последняя строка курсора (если BX = 1)

При возврате. Ничего не возвращается.

Замечания. Если выбран программный текстовый курсор (BX = 0), маски в CX и DX отображаются следующим образом:

Бит(ы)	Значение
0–7	код символа
8–10	цвет изображения
11	яркость
12–14	цвет фона
15	мерцание

Например, следующие значения определят программный курсор, который инвертирует цвета фона и изображения:

AX = 000AH
BX = 0000H
CX = 77FFH
DX = 7700H

Если выбран аппаратный текстовый курсор (BX = 1), значения в CX и DX представляют собой начальную и конечную строки развертки для мерцающего курсора, генерируемого видеоадаптером. Максимальное значение строки развертки, которое можно использовать, зависит от типа адаптера и текущего выдеорежима.

Int 33H
Функция 0BH(11)
Прочитать состояние счетчиков движения мыши

Возвращает чистое смещение мыши, произошедшее с момента последнего вызова этой функции. Возвращаемое значение измеряется в микки; положительное число указывает

на движение вправо или вниз, отрицательное число – на движение влево или вверх. Один микки представляет приблизительно 1/200 дюйма перемещения мыши.
При вызове. AX = 000BH
При возврате. CX = горизонтальный (X) счетчик микки
DX = вертикальный (Y) счетчик микки

Int 33H
Функция 0CH(12)
Установить определяемый пользователем обработчик событий мыши

Устанавливает маску события и адрес обработчика событий мыши прикладной программы.
При вызове. AX = 000CH
CX = маска события

Бит(ы)	**Значение** (если бит установлен)
0	движение мыши
1	левая клавиша нажата
2	левая клавиша отпущена
3	правая клавиша нажата
4	правая клавиша отпущена
5	средняя клавиша нажата
6	средняя клавиша отпущена
7–15	зарезервированы

ES:DX = сегмент:относительный_адрес обработчика
При возврате. Ничего не возвращается.
Замечания. Определяемый пользователем обработчик событий вызывается драйвером мыши с помощью дальнего вызова при следующем содержимом регистров:

AX Флаги событий мыши (см. маску события)
BX Состояние клавиш

Бит(ы)	**Значение** (если бит установлен)
0	левая клавиша нажата
1	правая клавиша нажата
2	средняя клавиша нажата
3–15	зарезервированы (0)

CX Горизонтальная (X) координата указателя
DX Вертикальная (Y) координата указателя
SI Вертикальный отсчет микки последней строки
DI Горизонтальный отсчет микки последней строки
DS Сегмент данных драйвера мыши

Если событие не генерирует вызов определяемого пользователем обработчика, потому что бит события не установлен в маске события, его характеристики по-прежнему отображаются в состоянии флагов события при вызовах обработчика в результате наступления разблокированных событий.

Вызовы обработчика блокируются с помощью прерывания Int 33H с функцией 00H или путем вызова настоящей функции с нулевой маской события.

См. также Int 33H с функциями 14H и 18H.

Int 33H
Функция 0DH(13)
Включить эмуляцию светового пера

Разблокирует эмуляцию светового пера драйвером мыши при работе с системой Бейсик фирмы IBM. Условие «перо опущено» создается путем одновременного нажатия и левой, и правой клавиш мыши.
При вызове. AX = 000DH
При возврате. Ничего не возвращается.

Int 33(H)
Функция 0EH(14)
Выключить эмуляцию светового пера

Блокирует эмуляцию светового пера драйвером мыши при работе с системой Бейсик фирмы IBM.
При вызове. AX = 000EH
При возврате. Ничего не возвращается.

Int 33H
Функция 0FH(15)
Установить отношение микки на пиксел

Устанавливает число микки на 8 пикселов при горизонтальном и вертикальном движении мыши. Один микки представляет приблизительно 1/200 дюйма перемещения мыши.
При вызове. AH = 000FH
CH = число микки по горизонтали (1 ... 32 767, по умолчанию = 8)
DX = число микки по вертикали (1 ... 32 767, по умолчанию = 16)
При возврате. Ничего не возвращается.

Int 33H
Функция 10H(16)
Установить область подавления указателя мыши

Определяет область подавления указателя мыши. Если указатель мыши лежит в указанной области, он не отображается на экране.
При вызове. AX = 0010H
CX = верхняя левая x-координата
DX = верхняя левая y-координата
SI = нижняя правая x-координата
DI = нижняя правая y-координата
При возврате. Ничего не возвращается.
Замечание. Заданная область подавления может быть заменена новым вызовом этой функции или отменена с помощью прерывания Int 33H с функциями 00H или 01H.

Int 33H
Функция 13H(19)
Установить порог удвоения скорости

Устанавливает пороговый предел удвоения скорости перемещения указателя на экране.

Действующее по умолчанию значение порога удвоения скорости составляет 64 микки/с.
При вызове. AX = 0013H
DX = пороговая скорость в микки/с
При возврате. Ничего не возвращается.
Замечание. Удвоение скорости движения указателя можно легко ликвидировать, установив очень большое значение пороговой скорости (например, 10000).

Int 33H
Функция 14H(20)
Сменить определяемые пользователем обработчики событий мыши

Устанавливает маску события и адрес обработчика событий мыши прикладной программы и возвращает маску события и адрес предыдущего обработчика. Вновь установленный обработчик вызывается драйвером мыши каждый раз при возникновении заданного события мыши.
При вызове. AX = 0014H
CX = маска события

Бит(ы)	**Значение** (если бит установлен)
0	движение мыши
1	левая клавиша нажата
2	левая клавиша отпущена
3	правая клавиша нажата
4	правая клавиша отпущена
5	средняя клавиша нажата
6	средняя клавиша отпущена
7–15	зарезервированы (0)

ES:DX = сегмент:относительный_адрес предыдущего обработчика
Замечания. В *Замечаниях* для прерывания Int 33H с функцией 0CH описывается информация, передаваемая определяемому пользователем обработчику событий. (См. также Int 33H с функцией 18H.)
Вызовы обработчика событий блокируются с помощью прерывания Int 33H с функцией 00H или путем установки нулевой маски события.

Int 33H
Функция 15H(21)
Получить размер буфера хранения состояния мыши

Возвращает размер буфера, требуемого для хранения текущего состояния драйвера мыши.
При вызове. AX = 0015H
При возврате. BX = размер буфера (в байтах).
Замечание. См. также Int 33H с функциями 16H и 17H.

Int 33H
Функция 16H(22)
Сохранить состояние драйвера мыши

Сохраняет состояние драйвера мыши в буфере пользователя.
Минимальный размер буфера следует предварительно определить с помощью прерывания Int 33H с функцией 15H.

При вызове. AX = 0016H
ES:DX = сегмент:относительный_адрес буфера
При возврате. Ничего не возвращается.
Замечание. Вызывайте эту функцию перед запуском дочерней программы с помощью прерывания Int 21H с функцией 4BH (EXEC), если дочерняя программа тоже использует мышь. После вызова EXEC восстановите старое состояние драйвера мыши с помощью прерывания Int 33H с функцией 17H.

Int 33H
Функция 17H(23)
Восстановить состояние драйвера мыши

Восстанавливает состояние драйвера мыши из буфера пользователя.
При вызове. AX = 0017H
ES:DX = сегмент:относительный_адрес буфера
При возврате. Ничего не возвращается.
Замечание. Состояние драйвера мыши должно быть предварительно сохранено в том же буфере с помощью прерывания Int 33H с функцией 16H. Формат данных в буфере не документируется и может изменяться.

Int 33H
Функция 18H(24)
Установить дополнительный обработчик событий мыши

Устанавливает адрес и маску события обработчика событий мыши прикладной программы. С помощью этой функции можно установить до трех обработчиков с различными масками событий. Когда возникает событие, соответствующее одной из масок, драйвер мыши вызывает соответствующий обработчик.
При вызове. AX = 0018H
CX = маска события

Бит(ы)	**Значение** (если бит установлен)
0	движение мыши
1	левая клавиша нажата
2	левая клавиша отпущена
3	правая клавиша нажата
4	правая клавиша отпущена
5	клавиша Shift нажата в процессе нажимания или отпускания клавиши мыши
6	клавиша Ctrl нажата в процессе нажимания или отпускания клавиши мыши
7	клавиша Alt нажата в процессе нажимания или отпускания клавиши мыши
8–15	зарезервированы (0)

ES:DX = сегмент:относительный_адрес обработчика
При возврате. Если функция выполнена успешно
AX = 0018H
Если функция не выполнена
AX = FFFFH
Замечания. При вызове этой функции по крайней мере один из битов 5,6 или 7 в регистре CX должен быть установлен.

Определенный пользователем обработчик событий вызывается драйвером мыши с помощью операции дальнего вызова при следующем содержимом регистров:
AX флаги событий мыши (см. маску события)
BX состояние клавиши

Бит(ы)	**Значение** (если бит установлен)
0	левая клавиша нажата
1	правая клавиша нажата
2	средняя клавиша нажата
3–15	зарезервированы (0)
CX	горизонтальная (X) координата указателя
DX	вертикальная (Y) координата указателя
SI	вертикальный отсчет микки последней строки
DI	горизонтальный отсчет микки последней строки
DS	сегмент данных драйвера мыши

Если событие не генерирует вызов определяемого пользователем обработчика, потому что бит события не установлен в маске события, его характеристики по-прежнему отображаются в состоянии флагов события при вызовах обработчика в результате наступления разблокированных событий.

Вызовы обработчика блокируются с помощью прерывания Int 33H с функцией 00H.

См. также Int 33H с функциями OCH и 14H.

Int 33H
Функция 19H(25)
Получить адрес дополнительного обработчика событий мыши

Возвращает адрес обработчика событий мыши, соответствующего заданной маске событий.

При вызове. AX = 0019H
CX = маска событий (см. Int 33H с функцией 18H)

При возврате. Если функция выполнена успешно
CX = маска событий
ES:DX = сегмент:относительный_адрес дополнительного обработчика событий
Если функция не выполнена (нет установленных обработчиков или маска событий не соответствует ни одному из установленных обработчиков)
CX = 0000H

Замечание. Прерывание Int 33H с функцией 18H позволяет установить до трех обработчиков событий с различными масками событий. С помощью настоящей функции можно найти обработчик, соответствующий конкретному событию, с целью его замены или блокировки.

Int 33H
Функция 1AH(26)
Установить чувствительность мыши

Устанавливает число микки на 8 пикселов для горизонтального и вертикального перемещений мыши, а также пороговую скорость удвоения скорости перемещения указателя на экране. Один микки представляет приблизительно 1/200 дюйма перемещения мыши.

При вызове. AX = 001AH
BX = число микки по горизонтали (1...32 767, по умолчанию = 8)
CX = число микки по вертикали (1...32 767, по умолчанию = 16)
DX = порог удвоения скорости в единицах микки/с (по умолчанию = 64)
При возврате. Ничего не возвращается.
Замечание. См. также Int 33H с функциями 0FH и 13H, с помощью которых можно раздельно установить отношение микки на пиксел и пороговую скорость, а также Int 33H с функцией 1BH, возвращающей текущие значения чувствительности.

Int 33H
Функция 1BH(27)
Получить чувствительность мыши

Возвращает текущее значение микки на пиксел для вертикального и горизонтального перемещений по экрану, а также пороговую скорость удвоения скорости перемещения указателя.
При вызове. AX = 001BH
При возврате. BX = число микки по горизонтали (1...32 767, по умолчанию = 8)
CX = число микки по вертикали (1...32 767, по умолчанию = 16)
DX = порог удвоения скорости в единицах микки/с (по умолчанию = 64)
Замечание. См. также Int 33H с функциями 0FH, 13H и 1AH.

Int 33H
Функция 1CH(28)
Установить частоту прерываний от мыши

Устанавливает частоту, с которой драйвер мыши опрашивает состояние мыши. Большая частота обеспечивает лучшее разрешение в графическом режиме, но может снизить производительность прикладной программы.
При вызове. AX = 001CH
BX = флаги частоты прерываний

Бит(ы)	Значение
0	прерывание запрещены
1	30 прерывание/с
2	50 прерывание/с
3	100 прерывание/с
4	200 прерывание/с
5–15	зарезервированы (0)

При возврате. Ничего не возвращается.
Замечания. Эта функция приложима только к мыши InPort.
Если в регистре BX установлено более одного бита, частота определяется младшим из них.

Int 33H
Функция 1DH(29)
Выбрать страницу указателя

Выбирает видеостраницу для указателя мыши.
При вызове. AX = 001FH
BX = страница

При возврате. Ничего не возвращается.
Замечание. Допустимые номера страниц зависят от текущего видеорежима. (См. Int 10H с функцией 05H.)

Int 33H
Функция 1EH(30)
Получить страницу указателя

Возвращает текущую видеостраницу для указателя мыши.
При вызове. AX = 001EH
При возврате. BX = страница

Int 33H
Функция 1FH(31)
Заблокировать драйвер мыши

Блокирует драйвер мыши и возвращает адрес предыдущего обработчика прерывания Int 33H.
При вызове. AX = 001FH
При возврате. Если функция выполнена успешно
AX = 001FH
ES:BX = сегмент:относительный_адрес обработчика прерывания Int 33H
Если функция не выполнена
AX = FFFFH
Замечания. При вызове этой функции драйвер мыши освобождает любые захваченные им векторы прерываний, кроме Int 33H (это могут быть векторы Int 10H, Int 71H и/или Int 74H). Прикладная программа может завершить процесс логического удаления драйвера мыши путем восстановления первоначального содержимого вектора Int 33H с помощью прерывания Int 21H с функцией 25H, используя адрес, возвращаемый этой функцией в регистрах ES:BX.

Int 33H
Функция 20H(32)
Разблокировать драйвер мыши

Разблокирует драйвер мыши и разрешает обработку прерываний от мыши.
При вызове. AX = 0020H
При возврате. Ничего не возвращается.
Замечание. См. также Int 33H с функцией 1FH.

Int 33H
Функция 21H(33)
Привести в исходное состояние драйвер мыши

Приводит драйвер мыши в исходное состояние и возвращает состояние драйвера. Если указатель мыши был видим, он удаляется с экрана и любые установленные ранее пользовательские обработчики событий мыши блокируются.
При вызове. AX = 0021H
При возврате. Если имеется поддержка мыши
AX = FFFFH

BX = число клавиш мыши
Если поддержка мыши отсутствует
AX = 0021H

Замечание. Эта функция отличается от прерывания Int 33H с функцией 00H в том отношении, что не происходит инициализации аппаратуры мыши.

Int 33H
Функция 22H(34)
Установить язык сообщений драйвера мыши

Выбирает язык, который будет использоваться драйвером мыши для запросов и сообщений об ошибках.
При вызове. AX = 0022H
BX = номер языка
0 = английский
1 = французский
2 = голландский
3 = немецкий
4 = шведский
5 = финский
6 = испанский
7 = португальский
8 = итальянский
При возврате. Ничего не возвращается.
Замечание. Эта функция реализована только в международных версиях драйвера мыши фирмы Microsoft.

Int 33N
Функция 23H(35)
Получить номер языка

Возвращает номер языка, используемого драйвером мыши для запросов и сообщений об ошибках.
При вызове. AX = 0023H
При возврате. BX = номер языка (см. Int 33H с функцией 22H).
Замечание. Эта функция реализована только в международных версиях драйвера мыши фирмы Microsoft.

Int 33H
Функция 24H(36)
Получить информацию о мыши

Возвращает номер версии драйвера мыши, тип мыши, а также номер запроса прерывания IRQ, используемого адаптером мыши.
При вызове. AX = 0024H
При возврате. BH = главный номер версии (6 для версии 6.10 и т. д.)
BL = дополнительный номер версии (0AH для версии 6.10 и т. д.)
CH = тип мыши
1 = мышь на магистрали
2 = последовательная мышь
3 = мышь InPort

4 = мышь PS/2
5 = мышь HP
CL = номер IRQ
0 = PS/2
2,3,4,5 или 7 = номер IRQ

Раздел 4. Справочник по функциям спецификации дополнительной памяти LOTUS/INTEL/MICROSOFT EMS

Пояснения для читателя

Спецификация дополнительной памяти (EMS) фирм Lotus/Intel/Microsoft определяет аппаратно-программную подсистему, совместимую с микрокомпьютерами, построенными на базе микропроцессоров 80 × 86 и работающими под управлением MS-DOS, которая позволяет прикладным программам обращаться к памяти с произвольным доступом объемом до 32 Мбайт с переключением банков. Программный компонент, носящий название диспетчера дополнительной памяти (Expanded Memory Manager, EMM), устанавливается при инициализации системы с помощью директивы DEVICE = в файле CONFIG.SYS, располагающемся в корневом каталоге загружаемого диска.

Прикладная программа, удостоверившись, что EMM присутствует, непосредственно взаимодействует с EMM с помощью программного прерывания 67H. Конкретная функция EMM определяется значением в регистре AH, а состояние успешного выполнения или ошибки возвращается также в регистре AH (коды ошибок приведены на с. 165–166). Другие параметры и результаты передаются или возвращаются в регистрах или буферах.

Условные обозначения в заголовках описаний функций показывают, в какой версии EMS была введена поддержка этой функции. Функцией можно пользоваться и во всех последующих версиях EMS, если только явным образом не указано обратное.

Условные обозначения версий, используемые в разделах описаний, параметров, результатов или замечаний, определяют конкретный номер (старший или младший) версии EMS. Знак +, включенный в условное обозначение, определяет указанную и все последующие версии.

Материал настоящего раздела был сверен со спецификацией дополнительной памяти версии 4.0, датированной октябрем 1987 г. (номер документа Intel 300275–005). Этот документ распространяется корпорацией Intel: Intel Corporation, 5200 N. E. Elam Young Parkway, Hillsboro, OR 97124.

Список функций EMM

Функция	Подфункция	Описание
40H(64)		Получить состояние
41H(65)		Получить адрес кадра страниц
42H(66)		Получить число страниц
43H(67)		Выделить дескриптор и страницы
44H(68)		Отобразить страницу дополнительной памяти
45H(69)		Освободить дескриптор и дополнительную память
46H(70)		Получить версию
47H(71)		Сохранить отображение страниц
48H(72)		Восстановить отображение страниц
49H(73)		Зарезервирована
4AH(74)		Зарезервирована
4BH(75)		Получить число дескрипторов

Продолжение таблицы

Функция	Подфункция	Описание
4CH(76)		Получить страницы дескриптора
4DH(77)		Получить страницы всех дескрипторов
4EH(78)	00H	Сохранить отображение страниц
4EH(78)	01H	Восстановить отображение страниц
4EH(78)	02H	Сохранить и восстановить отображение страниц
4EH(78)	03H	Получить размер информации об отображении страниц
4FH(79)	00H	Сохранить частичное отображение страниц
4FH(79)	01H	Восстановить частичное отображение страниц
4FH(79)	02H	Получить размер информации о частичном отображении страниц
50H(80)	00H	Отобразить несколько страниц по номеру
50H(80)	01H	Отобразить несколько страниц по адресу
51H(81)		Переназначить страницы дескриптору
52H(82)	00H	Получить атрибут дескриптора
52H(82)	01H	Установить атрибут дескриптора
52H(82)	02H	Получить возможности атрибута
53H(83)	00H	Получить имя дескриптора
53H(83)	01H	Установить имя дескриптора
54H(84)	00H	Получить имена всех дескрипторов
54H(84)	01H	Поиск имени дескриптора
54H(84)	02H	Получить полное число дескрипторов
55H(85)	00H	Отобразить страницы по номеру и осуществить переход
55H(85)	01H	Отобразить страницы по адресу и осуществить переход
56H(86)	00H	Отобразить страницы по номеру и вызвать подпрограмму
56H(86)	01H	Отобразить страницы по адресу и вызывать подпрограмму
56H(86)	02H	Получить стековое пространство для отображения страниц и вызвать подпрограмму
57H(87)	00H	Переместить район памяти
57H(87)	01H	Обмен районов памяти
58H(88)	00H	Получить адреса отображаемых страниц
58H(88)	01H	Получить число отображаемых страниц
59H(89)	00H	Получить аппаратную конфигурацию
59H(89)	01H	Получить число исходных страниц
5AH(90)	00H	Выделить дескриптор и стандартные страницы
5AH(90)	01H	Выделить дескриптор и исходные страницы
5BH(91)	00H	Получить дополнительные регистры отображения
5BH(91)	01H	Установить дополнительные регистры отображения
5BH(91)	02H	Получить размер области сохранения дополнительных регистров отображения
5BH(91)	03H	Выделить дополнительный набор регистров отображения
5BH(91)	04H	Освободить дополнительный набор регистров отображения
5BH(91)	05H	Выделить набор регистров ПДП
5BH(91)	06H	Разрешить ПДП через дополнительный набор регистров отображения
5BH(91)	07H	Запретить ПДП через дополнительный набор регистров отображения
5BH(91)	08H	Освободить набор регистров ПДП
5CH(92)		Подготовить диспетчер дополнительной памяти к теплой перезагрузке
5DH(93)	00H	Разрешить системные функции EMM
5DH(93)	01H	Запретить системные функции EMM
5DH(93)	02H	Освободить ключ доступа

Int 67H [EMS 3.0]
Функция 40H(64)
Получить состояние

Возвращает код состояния, указывающий на наличие и функционирование аппаратного и программного обеспечения дополнительной памяти.
При вызове. AH = 40H
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечание. Этот вызов следует использовать только после того, как прикладная программа установила, что диспетчер дополнительной памяти фактически присутствует. Это можно сделать с помощью одной из методик, описанных в гл. 11.

Int 67H [EMS 3.0]
Функция 41H(65)
Получить адрес кадра страниц

Возвращает адрес сегмента кадра страниц, используемого диспетчером дополнительной памяти.
При вызове. AH = 41H
При возврате. Если функция выполнена успешно
AH = 00H
BX = база сегмента кадра страниц
Если функция не выполнена
AH = код ошибки
Замечания. Кадр страниц разделен на четыре страницы по 16 Кбайт, используемые для отображения логических страниц дополнительной памяти на физическое адресное пространство ЦП. Для использования этой функции прикладной программе не требуется обладать дескриптором EMM.

[EMS 4.0] Отображение страниц дополнительной памяти не обязательно ограничивается кадром страниц размером 16 Кбайт. (См. также Int 67H с функцией 58H, подфункцией 00H.)

Int 67H [EMS 3.0]
Функция 42H(66)
Получить число страниц

Позволяет получить полное число наличных логических страниц дополнительной памяти в системе и число еще невыделенных страниц.
При вызове. AH = 42H
При возврате. Если функция выполнена успешно
AH = 00H
BX = число невыделенных страниц
DX = полное число страниц
Если функция не выполнена
AH = код ошибки
Замечания. Для использования этой функции прикладной программе не требуется обладать дескриптором EMM.

[EMS 4.0] См. также Int 67H с функцией 59H и подфункцией 01H.

Int 67H **[EMS 3.0]**
Функция 43H(67)
Выделить дескриптор и страницы

Позволяет получить дескриптор EMM и выделяет логические страницы дополнительной памяти, которые будут управляться этим дескриптором.
При вызове. AH = 43H
BX = число выделенных страниц (должно быть не 0)
При возврате. Если функция выполнена успешно
AH = 00H
DX = дескриптор EMM
Если функция не выполнена
AH = код ошибки
Замечания. Функция является эквивалентом функции открытия файла для диспетчера дополнительной памяти. Возвращаемый дескриптор является аналогом дескриптора файла; он владеет определенным числом страниц дополнительной памяти. Дескриптор следует использовать с каждым последующим запросом на отображение памяти; его следует освободить с помощью операции закрытия перед завершением прикладной программы.

Функция может не выполниться, если не осталось дескрипторов, которые можно было бы выделить, или если числá страниц дополнительной памяти не хватает для удовлетворения запроса. В последнем случае для определения фактического числа доступных страниц можно использовать прерывание Int 67H с функцией 42H.

[EMS 4.] Для изменения числа страниц, выделенных дескриптору EMM, можно вызвать прерывание Int 67H с функцией 51H.

[EMS 4.0] Страницы, выделенные с помощью этой функции, всегда имеют размер 16 Кбайт для совместимости с ранними версиями EMS. (См. также Int 67H с функцией 5AH и подфункциями 00H и 01H.)

[EMS 4.0] Дескриптор 0000H всегда доступен для использования операционной системой, и предварительный вызов этой функции не требуется. Для назначения зарезервированному дескриптору требуемого числа страниц операционная система должна вызвать прерывание Int 67H с функцией 51H.

Int 67H **[EMS 3.0]**
Функция 44H(68)
Отобразить страницу дополнительной памяти

Отображает одну из логических страниц дополнительной памяти, назначенных дескриптору, на страницу физической памяти, доступную ЦП.
При вызове. AH = 44H
AL = физическая страница
BX = логическая страница
DX = дескриптор EMM
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечания. Номер логической страницы находится в диапазоне 0 ... n − 1, где n − число страниц, выделенных дескриптору (в том числе заново) предыдущим вызовом Int 67H с функциями 43H, 51H или 5AH. Логические страницы, выделяемые с помощью прерывания Int 67H с функцией 43H или с функцией 5AH и подфункцией 00H, всегда

имеют размер 16 Кбайт; логические страницы, выделяемые с помощью прерывания Int 67H с функцией 5AH и подфункцией 01H, называются исходными страницами (raw pages) и не обязательно имеют размер 16 Кбайт.

[EMS 3] Физическая страница находится в диапазоне 0...3 и лежит внутри кадра страницы EMM, базовый адрес которого можно получить с помощью прерывания Int 67H с функцией 41H.

[EMS 4.0] Список доступных физических страниц и их адресов можно получить с помощью прерывания Int 67H с функцией 58H и подфункцией 00H.

[EMS 4.0] Если при вызове этой функции BX = −1, снимается отображение указанной физической страницы (т. е. страница становится недоступной для чтения или записи).

Int 67H **[EMS 3.0]**
Функция 45H(69)
Освободить дескриптор и дополнительную память

Отменяет выделение страниц дополнительной памяти, назначенных дескриптору, после чего освобождает дескриптор.
При вызове. AH = 45H
DX = дескриптор EMM
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечания. Если эта функция не вызвана, а программа завершилась, страницы EMS, которыми обладала программа, остаются недоступными до перезапуска системы. Программы, использующие EMS, должны устанавливать собственные обработчики Ctrl-C и критических ошибок (прерывания Int 23H и Int 24H) во избежание неожиданного завершения.

[EMS 4.0] При освобождении дескриптора его имя заполняется нулями ASCII.

Int 67H **[EMS 3.0]**
Функция 46H(70)
Получить версию

Возвращает версию EMS, поддерживаемую диспетчером дополнительной памяти.
При вызове. AH = 46H
При возврате. Если функция выполнена успешно
AH = 00H
AL = номер версии
Если функция не выполнена
AH = код ошибки
Замечания. Номер версии возвращается в двоично-десятичном формате, причем целая часть находится в старших четырех битах AL, а дробная – в младших четырех битах. Например, если EMM поддерживает версию 3.2 EMS, значение, возвращаемое в регистре AL, составляет 32H.

Прикладные программы обязательно должны проверять номер версии EMM, чтобы убедиться, что все требуемые ими функции EMM доступны.

Int 67H **[EMS 3.0]**
Функция 47H(71)
Сохранить отображение страниц

Сохраняет содержимое регистров отображения страниц аппаратуры дополнительной памяти, связывая это содержимое с конкретным дескриптором EMM.
При вызове. AH = 47H
DX = дескриптор EMM
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечания. Эта функция используется обработчиками прерываний и драйверами устройств, которые должны иметь доступ к дополнительной памяти. Дескриптор EMM, предоставляемый этой функции, является тем дескриптором, который был назначен не прерываемой программе, а обработчику или драйверу в процессе выполнения его инициализирующей последовательности.

Контекст отображения восстанавливается последующим вызовом прерывания Int 67H с функцией 48H.

[EMS 4.0] Эта функция сохраняет только состояние отображения для кадра страниц размером 64 Кбайт, определенного в EMS 3.0. Программы, реализующие дополнительные возможности EMS 4.0, должны вместо этой функции использовать прерывание Int 67H с функциями 4EH или 4FH.

Int 67H **[EMS 3.0]**
Функция 48H(72)
Восстановить отображение страниц

Восстанавливает содержимое регистров отображения страниц аппаратуры дополнительной памяти, используя значения, связанные с указанным дескриптором предыдущим вызовом прерывания Int 67H с функцией 47H.
При вызове. AH = 48H
DX = дескриптор EMM
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечания. Эта функция используется обработчиками прерываний и драйверами устройств, которые должны иметь доступ к дополнительной памяти. Дескриптор EMM, предоставляемый этой функции, является тем дескриптором, который был назначен не прерываемой программе, а обработчику или драйверу в процессе выполнения его инициализирующей последовательности.

[EMS 4.0] Эта функция восстанавливает только состояние отображения для кадра страниц размером 64 Кбайт, определенного в EMS 3.0. Программы, реализующие дополнительные возможности, должны вместо этой функции использовать прерывание Int 67H с функциями 4EH и 4FH.

Int 67H **[EMS 3.0]**
Функция 49H(73)
Зарезервирована

Эта функция была определена в EMS версии 3.0, но не документирована в более поздних версиях EMS, так что ею не следует пользоваться в прикладных программах.

Int 67H **[EMS 3.0]**
Функция 4AH(74)
Зарезервирована

Эта функция была определена в EMS версии 3.0, но не документирована в более поздних версиях EMS, так что ею не следует пользоваться в прикладных программах.

Int 67H **[EMS 3.0]**
Функция 4BH(75)
Получить число дескрипторов

Возвращает число активных дескрипторов дополнительной памяти.
При вызове. AH = 4BH
При возврате. Если функция выполнена успешно
AH = 00H
BX = число активных дескрипторов EMM
Если функция не выполнена
AH = код ошибки
Замечания. Если возвращенное число дескрипторов EMM равно 0, диспетчер дополнительной памяти бездействует и дополнительная память не используется.

Число, возвращаемое этой функцией, не обязательно совпадает с числом программ, использующих дополнительную память, потому что одна программа может владеть несколькими дескрипторами EMM.

Int 67H **[EMS 3.0]**
Функция 4CH(76)
Получить страницы дескриптора

Возвращает число страниц дополнительной памяти, выделенных указанному дескриптору EMM.
При вызове. AH = 4CH
DX = дескриптор EMM
При возврате. Если функция выполнена успешно
AH = 00H
BX = число страниц EMM
Если функция не выполнена
AH = код ошибки
Замечания. [EMS 3.0] Полное число страниц, выделенных дескриптору, никогда не превышает 512. Дескриптору никогда не выделяются нулевые страницы.

[EMS 4.0] Полное число страниц, выделенных дескриптору, никогда не превышает 2048. Дескриптору могут выделяться нулевые страницы дополнительной памяти.

Int 67H **[EMS 3.0]**
Функция 4DH(77)
Получить страницы всех дескрипторов

Возвращает массив, содержащий все активные дескрипторы и число страниц дополнительной памяти, связанных с каждым дескриптором.
При вызове. AH = 4DH
ES:DI = сегмент:относительный_адрес буфера (см. *Замечания*)
При возврате. Если функция выполнена успешно
AH = 00H
BX = число активных дескрипторов EMM
и буфер, заполненный, как описано в *Замечаниях*
Если функция не выполнена
AH = код ошибки
Замечания. Буфер заполняется последовательностью записей DWORD (32 бит), по одной на каждый активный дескриптор EMM. Первое слово записи содержит дескриптор, а второе слово содержит число страниц, выделенных этому дескриптору.

Максимальное число активных дескрипторов составляет 256 (включая дескриптор 0 операционной системы), так что буфер размером 1024 байт достаточен во всех случаях.

Int 67H **[EMS 3.2]**
Функция 4EH(78), подфункция 00H
Сохранить отображение страниц

Сохраняет в указанном буфере текущее состояние отображения страниц аппаратуры дополнительной памяти.
При вызове. AH = 4EH
AL = 00H
ES:DI = сегмент:относительный_адрес буфера (см. *Замечания*)
При возврате. Если функция выполнена успешно
AH = 00H
и буфер, заполненный информацией об отображении (см. *Замечания*)
Если функция не выполнена
AH = код ошибки
Замечания. Буфер получает информацию, необходимую для восстановления состояния регистров отображения с помощью прерывания Int 67H с функцией 4EH и подфункцией 01H. Формат информации может варьироваться.

Размер буфера, требуемый для выполнения этой функции, можно определить с помощью прерывания Int 67H с функцией 4EH и подфункцией 03H.

В отличие от Int 67H с функцией 47H эта функция не требует дескриптора.

Int 67H **[EMS 3.2]**
Функция 4EH(78), подфункция 01H
Восстановить отображение страниц

Восстанавливает состояние отображения страниц аппаратуры дополнительной памяти, используя информацию из указанного буфера.
При вызове. AH = 4EH
AL = 01H
DS:SI = сегмент:относительный_адрес буфера (см. *Замечания*)

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечания. Буфер содержит информацию, необходимую для восстановления состояния регистров отображения после предшествующего вызова Int 67H с функцией 4EH и подфункцией 00H или 02H. Формат информации может варьироваться.

В отличие от Int 67H с функцией 48H эта функция не требует дескриптора.

Int 67H [EMS 3.2]
Функция 4EH(78), подфункция 02H
Сохранить и восстановить отображение страниц

Сохраняет в буфере текущее состояние отображения страниц аппаратуры дополнительной памяти, а затем устанавливает состояние отображения, используя информацию из другого буфера.

При вызове. AH = 4EH
AL = 02H
DS:SI = сегмент:относительный_адрес буфера, содержащего информацию об отображении (см. *Замечания*)
ES:DI = сегмент:относительный_адрес буфера, принимающего информацию об отображении (см. *Замечания*)

При возврате. Если функция выполнена успешно
AH = 00H
и буфер, заполненный информацией об отображении, на который указывают регистры ES:DI (см. *Замечания*)
Если функция не выполнена
AH = код ошибки

Замечания. Буфер, адресуемый через DS:SI, содержит информацию, необходимую для восстановления состояния регистров отображения после предшествующего вызова Int 67H с функцией 4EH и подфункцией 00H или 02H. Формат информации может варьироваться.

Размеры буферов, требуемых для выполнения этой функции, можно определить с помощью прерывания Int 67H с функцией 4EH и подфункцией 03H.

В отличие от Int 67H с функциями 47H и 48H эта функция не требует дескриптора.

Int 67H [EMS 3.2]
Функция 4EH(78), подфункция 03H
Получить размер информации об отображении страниц

Возвращает размер буфера, требуемого для получения информации об отображении страниц с помощью прерывания Int 67H с функцией 4EH и подфункциями 00H и 02H.

При вызове. AH = 4EH
AL = 03H

При возврате. Если функция выполнена успешно
AH = 00H
AL = размер буфера (в байтах)
Если функция не выполнена
AH = код ошибки

Int 67H **[EMS 4.0]**
Функция 4FH (79), подфункция 00H
Сохранить частичное отображение страниц

Сохраняет в указанном буфере состояние подмножества регистров отображения страниц дополнительной памяти.

При вызове. AH = 4FH
AL = 00H
DS:SI = сегмент:относительный_адрес карты отображения
ES:DI = сегмент:относительный_адрес буфера, принимающего состояние отображения (см. *Замечания*)

При возврате. Если функция выполнена успешно
AH = 00H
и буфер, заполненный информацией об отображении (см. *Замечания*)
Если функция не выполнена
AH = код ошибки

Замечания. Карта отображения содержит число отображаемых сегментов, за которым следуют адреса сегментов отображаемых районов памяти (один сегмент на слово).

Для определения размера буфера, требуемого для хранения состояния отображения, используйте прерывание Int 67H с функцией 4FH и подфункцией 02H.

Int 67H **[EMS 4.0]**
Функция 4FH(79), подфункция 01H
Восстановить частичное отображение страниц

Восстанавливает состояние подмножества регистров отображения страниц дополнительной памяти.

При вызове. AH = 4FH
AL = 01H
DS:SI = сегмент:относительный_адрес буфера (см. *Замечание*)

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечание. Буфер содержит информацию об отображении и должен быть уже подготовлен предшествующим вызовом Int 67H с функцией 4FH и подфункцией 00H.

Int 67H **[EMS 4.0]**
Функция 4FH(79), подфункция 02H
Получить размер информации о частичном отображении страниц

Возвращает размер буфера, который потребуется для получения информации о частичном отображении страниц с помощью прерывания Int 67H с функцией 4FH и подфункцией 00H.

При вызове. AH = 4FH
AL = 02H
BX = число страниц

При возврате. Если функция выполнена успешно
AH = 00H
AL = размер массива (в байтах)
Если функция не выполнена
AH = код ошибки

Int 67H **[EMS 4.0]**
Функция 50H(80), подфункция 00H
Отобразить несколько страниц по номеру

Отображает одну или более логических страниц дополнительной памяти, назначенных дескриптору, на физические страницы памяти, доступные ЦП. Ссылка на физические страницы осуществляется по их номерам.

При вызове. AH = 50H
AL = 00H
CX = число отображаемых страниц
DX = дескриптор EMM
DS:SI = сегмент:относительный_адрес буфера (см. *Замечание*)

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечание. Буфер содержит последовательность двухсловных записей (32 бит), управляющих отображаемыми страницами. Первое слово каждой записи содержит номер логической страницы дополнительной памяти; второе слово содержит номер физической страницы, на которую осуществляется отображение. Если логическая страница = −1, отображение физической страницы снимается (страница становится недоступной для чтения или записи).

Int 67H **[EMS 4.0]**
Функция 50H(80), подфункция 01H
Отобразить несколько страниц по адресу

Отображает одну или более логических страниц дополнительной памяти, назначенных дескриптору, на физические страницы памяти, доступные ЦП. Ссылка на физические страницы осуществляется по адресам их сегментов.

При вызове. AH = 50H
AL = 01H
CX = число отображаемых страниц
DX = дескриптор EMM
DS:SI = сегмент:_относительный адрес буфера (см. *Замечания*)

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечания. Буфер содержит последовательность двухсловных записей (32 бит), управляющих отображаемыми страницами. Первое слово каждой записи содержит номер логической страницы; второе слово содержит адрес сегмента физической страницы, на которую осуществляется отображение. Если логическая страница = −1, отображение физической страницы снимается (страница становится недоступной для чтения или записи).

Адреса отображаемых сегментов могут быть получены путем вызова Int 67H с функцией 58H и подфункцией 00H.

Int 67H **[EMS 4.0]**
Функция 51H(81)
Переназначить страницы дескриптору

Изменяется число страниц дополнительной памяти, выделенных дескриптору EMM.
При вызове. AH = 51H
BX = новое число страниц
DX = дескриптор EMM
При возврате. Если функция выполнена успешно
AH = 00H
BX = логические страницы, которыми обладает дескриптор EMM
Если функция не выполнена
AH = код ошибки

Замечание. Если число запрашиваемых страниц = 0, дескриптор остается активным и страницы могут быть переназначены дескриптору позже; поэтому перед завершением прикладной программы дескриптор должен быть освобожден с помощью прерывания Int 67H с функцией 45H.

Int 67H **[EMS 4.0]**
Функция 52H(82), подфункция 00H
Получить атрибут дескриптора

Возвращает атрибут (сохраняемый или несохраняемый), связанный с указанным дескриптором. Сохраняемый дескриптор памяти и содержимое страниц дополнительной памяти, выделенных этому дескриптору, не изменяются при выполнении теплой перезагрузки (перезапуск системы с помощью Ctrl-Alt-Del).
При вызове. AH = 52H
AL = 00H
DX = дескриптор EMM
При возврате. Если функция выполнена успешно
AH = 00H
AL = атрибут
0 = несохраняемый
1 = сохраняемый
Если функция не выполнена
AH = код ошибки

Int 67H **[EMS 4.0]**
Функция 52H(82), подфункция 01H
Установить атрибут дескриптора

Устанавливает атрибут (сохраняемый или несохраняемый), связанный с указанным дескриптором. Сохраняемый дескриптор памяти и содержимое страниц дополнительной памяти, выделенных этому дескриптору, не изменяются при выполнении теплой перезагрузки (перезапуска системы с помощью Ctrl-Alt-Del).
При вызове. AH = 52H
AL = 01H
0 = несохраняемый
1 = сохраняемый
DX = дескриптор EMM
При возврате. Если функция выполнена успешно

AH = 00H
Если функция не выполнена
AH = код ошибки

Замечание. Если аппаратура дополнительной памяти не поддерживает сохраняемые страницы, эта функция возвращает ошибку.

Int 67H **[EMS 4.0]**
Функция 52H(82), подфункция 02H
Получить возможности атрибута

Возвращает код, указывающий, поддерживается ли диспетчером дополнительной памяти и аппаратурой атрибут сохраняемости дескриптора EMM.

При вызове. AH = 00H
AL = 02H

При возврате. Если функция выполнена успешно
AH = 00H
AL = возможности атрибута
0 = поддерживаются только несохраняемые дескрипторы
1 = поддерживаются как несохраняемые, так и сохраняемые дескрипторы
Если функция не выполнена
AH = код ошибки

Int 67H **[EMS 4.0]**
Функция 53H(83), подфункция 00H
Получить имя дескриптора

Возвращает 8-символьное имя, назначенное дескриптору.

При вызове. AH = 53H
AL = 00H
DL = дескриптор EMM
ES:DI = сегмент:относительный_адрес 8-байт буфера

При возврате. Если функция выполнена успешно
AH = 00H
и имя дескриптора в указанном буфере
Если функция не выполнена
AH = код ошибки

Замечание. Имя дескриптора при выделении или отмене выделения инициализируется восемью нулевыми байтами. Активному дескриптору может быть назначено другое имя с помощью прерывания Int 67H с функцией 53H и подфункцией 01H. Байты в имени дескриптора не обязательно должны быть ASCII-символами.

Int 67H **[EMS 4.0]**
Функция 53H(83), подфункция 01H
Установить имя дескриптора

Назначает имя дескриптору EMM.

При вызове. AH = 53H
AL = 01H
DX-дескриптор EMM
DS:SI = сегмент:относительный_адрес 8-байт имени

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечания. Байты в имени дескриптора не обязательно должны быть ASCII-символами, но последовательность восьми нулевых байтов зарезервирована для безымянного дескриптора (состояние, действующее по умолчанию после выделения или отмены выделения дескриптора). При необходимости имя дескриптора дополняется до длины 8 байт нулевыми байтами.

Дескриптор можно переименовать в любое время.

При включении системы имена всех дескрипторов инициализируются восемью нулевыми байтами. Имя сохраняемого дескриптора сохраняется при теплой перезагрузке. (См. Int 67H с функцией 52H и подфункциями 00H и 02H.)

Int 67H **[EMS 4.0]**
Функция 54H(84), подфункция 00H
Получить имена всех дескрипторов

Возвращает имена всех активных дескрипторов.
При вызове. AH = 54H
AL = 00H
ES:DI = сегмент:относительный адрес буфера (см. *Замечания*)
При возврате. Если функция выполнена успешно
AH = 00H
AL = число активных дескрипторов
и буфер, заполненный информацией об именах дескрипторов (см. *Замечания*)
Если функция не выполнена
AH = код ошибки

Замечания. Функция заполняет буфер последовательностью 10-байт записей. Первые 2 байт каждой записи содержат дескриптор EMM; следующие 8 байт содержат имя, связанное с дескриптором. Дескрипторы, которым не было назначено имя, имеют в качестве имени 8 нулевых байтов.

Поскольку число активных дескрипторов не может превышать 255, длину буфера можно ограничить величиной 2550 байт.

Int 67H **[EMS 4.0]**
Функция 54H(84), подфункция 01H
Поиск имени дескриптора

Возвращает дескриптор EMM, связанный с указанным именем.
При вызове. AH = 54H
AL = 01H
DS:SI = сегмент:относительный_адрес 8-байт имени дескриптора
При возврате. Если функция выполнена успешно
AH = 00H
DX = дескриптор EMM
Если функция не выполнена
AH = код ошибки

Int 67H **[EMS 4.0]**
Функция 54H(84), подфункция 02H
Получить полное число дескрипторов

Возвращает полное число дескрипторов, поддерживаемых диспетчером дополнительной памяти, включая дескриптор операционной системы (0).
При вызове. AH = 54H
AL = 02H
При возврате. Если функция выполнена успешно
AH = 00H
BX = число дескрипторов
Если функция не выполнена
AH = код ошибки

Int 67H **[EMS 4.0]**
Функция 55H(85), подфункции 00H и 01H
Отобразить страницы и осуществить переход

Изменяет контекст отображения дополнительной памяти и передает управление по указанному адресу.
При вызове. AH = 55H
AL = подфункция
0 = отображение с использованием номеров физических страниц
1 = отображение с использованием сегментов физических страниц
DX = дескриптор EMM
DS:SI = сегмент:относительный_адрес буфера (см. *Замечания*)
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечания. Буфер содержит записи отображения и перехода в следующем формате:

Смещение	Длина	Описание
00H	4	Дальний указатель на точку перехода
04H	1	Число страниц, требующих отображения перед переходом
05H	4	Дальний указатель на карту отображения (см. ниже)

Карта отображения в свою очередь состоит из двухсловных записей (32 бит), по одной на страницу. Первое слово каждой записи содержит номер логической страницы; второе слово содержит номер физической страницы или сегмента (в зависимости от значения в регистре AL), на которые выполняется отображение.

Запрос на отображение нулевого количества страниц не рассматривается как ошибка; результатом является просто дальний переход.

Int 67H **[EMS 4.0]**
Функция 56H(86), подфункции 00H и 01H
Отобразить страницы и вызвать подпрограмму

Изменяет контекст отображения дополнительной памяти и выполняет дальний вызов подпрограммы по указанному адресу. Когда вызванная подпрограмма выполняет команду дальнего возврата, EMM снова изменяет контекст отображения страниц

в соответствии с указаниями, а затем возвращает управление в исходную вызывающую программу.

При вызове. AH = 56H
AL = подфункция
0 = отображение с использованием номеров физических страниц
1 = отображение с использованием сегментов физических страниц
DX = диспетчер EMM
DS:SI = сегмент:относительный_ адрес буфера (см. *Замечания*)

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечания. Формат буфера, содержащего информацию об отображении и вызове подпрограммы, следующий:

Смещение	Длина	Описание
00H	4	Дальний указатель на вызываемую подпрограмму
04H	1	Число страниц, отображаемых перед вызовом подпрограммы
05H	4	Дальний указатель карты страниц, отображаемых перед вызовом подпрограммы (см. ниже)
09H	1	Число страниц, отображаемых перед возвратом
0AH	4	Дальний указатель карты страниц, отображаемых перед возвратом (см. ниже)
0EH	8	Зарезервированы (0)

Обе карты отображения имеют одинаковый формат и состоят из последовательностей двухсловных записей, по одной на страницу. Первое слово каждой записи содержит номер логической страницы; второе слово содержит номер физической страницы или сегмента (в зависимости от значения в регистре AL), на которые выполняется отображение.

Запрос на отображение нулевого количества страниц не является ошибкой; результатом является просто дальний вызов подпрограммы.

Функция использует дополнительное пространство стека для сохранения информации о контексте отображения; требуемый объем стекового пространства можно определить вызовом Int 67H с функцией 56H и подфункцией 02H.

Int 67H **[EMS 4.0]**
Функция 56H(86), подфункция 02H
Получить стековое пространство для отображения страниц и вызвать подпрограмму

Возвращает число байтов стекового пространства, требуемого для выполнения прерывания Int 67H с функцией 56H и подфункциями 00H и 01H.

При вызове. AH = 56H
AL = 02H

При возврате. Если функция выполнена успешно
AH = 00H
BX = требуемое стековое пространство (в байтах)
Если функция не выполнена
AH = код ошибки

Int 67H **[EMS 4.0]**
Функция 57H(87), подфункция 00H
Переместить район памяти

Копирует район памяти из одного места в обычной или дополнительной памяти в любое другое место без нарушения контекста текущего отображения дополнительной памяти.

При вызове. AH = 57H
AL = 00H
DS:SI = сегмент:относительный_адрес буфера (см. *Замечания*)

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечания. Формат буфера, управляющего операцией перемещения, следующий:

Смещение	Длина	Описание
00H	4	Длина района в байтах
04H	1	Тип памяти-источника (0 = обычная, 1 = дополнительная)
05H	2	Дескриптор памяти-источника
07H	2	Относительный адрес памяти-источника
09H	2	Сегмент или номер физической страницы памяти-источника
0BH	1	Тип памяти-приемника (0 = обычная, 1 = дополнительная)
0CH	2	Дескриптор памяти-приемника
0EH	2	Относительный адрес памяти-приемника
10H	2	Сегмент или номер физической страницы памяти-приемника

Длина 0 байт не является ошибкой. Максимальная длина пересылки составляет 1 Мбайт. Если длина превышает одну страницу дополнительной памяти, для поставки или приема данных используются последующие страницы дополнительной памяти (столько, сколько требуется).

Если адреса источника и приемника перекрываются, пересылка осуществляется таким образом, что приемник получает неповрежденную копию первоначальных данных, и возвращается ненулевое состояние.

Int 67H **[EMS 4.0]**
Функция 57H(87), подфункция 01H
Обмен районов памяти

Обменивает любые два района памяти в обычной или дополнительной памяти без нарушения контекста текущего отображения дополнительной памяти.

При вызове. AH = 57H
AL = 01H
DS:SI = сегмент:относительный_адрес буфера (см. *Замечания*)

При возврате. Если функция выполнена успешно
AH = 00H

Если функция не выполнена
AH = код ошибки

Замечания. Формат буфера, управляющего операцией обмена, такой же, как для прерывания Int 67H с функцией 57H и подфункцией 00H.

Обмен нулевым числом байтов не является ошибкой. Максимальная длина обмена составляет 1 Мбайт. Если длина превышает одну страницу дополнительной памяти, для постановки или приема данных используются последующие страницы дополнительной памяти (столько, сколько требуется).

Если адреса источника и приемника перекрываются, обмен не выполняется и возвращается ошибка.

Int 67H **[EMS 4.0]**
Функция 58H(88), подфункция 00H
Получить адреса отображаемых страниц

Возвращает адрес базы сегмента и номер физической страницы для каждой отображаемой страницы в системе.

При вызове. AH = 58H
AL = 00H
ES:DI = сегмент:относительный_адрес буфера (см. *Замечания*)

При возврате. Если функция выполнена успешно
AH = 00H
CX = число записей в массиве отображаемых физических страниц
и информация о номере страницы и адресе в буфере (см. *Замечания*)
Если функция не выполнена
AH = код ошибки

Замечания. После возврата из прерывания буфер содержит последовательность двухсловных записей, по одной на каждую отображаемую страницу. Первое слово записи содержит адрес базы сегмента страницы; второе слово содержит номер физической страницы. Записи упорядочены в порядке возрастания адресов сегментов.

Размер требуемого буфера можно вычислить с помощью информации, возвращаемой после прерывания Int 67H с функцией 58H и подфункцией 01H.

Int 67H **[EMS 4.0]**
Функция 58H(88), подфункция 01H
Получить число отображаемых страниц

Возвращает число отображаемых физических страниц.

При вызове. AH = 58H
AL = 01H

При возврате. Если функция выполнена успешно
AH = 00H
CX = число отображаемых физических страниц
Если функция не выполнена
AH = код ошибки

Замечание. Информацию, возвращаемую этой функцией, можно использовать для вычисления размера буфера, требуемого для выполнения прерывания Int 67H с функцией 58H и подфункцией 00H.

Int 67H **[EMS 4.0]**
Функция 59H(89), подфункция 00H
Получить аппаратную конфигурацию

Возвращает информацию о конфигурации аппаратуры дополнительной памяти.
При вызове. AH = 59H
AL = 00H
ES:DI = сегмент:относительный адрес буфера (см. *Замечания*)
При возврате. Если функция выполнена успешно
AH = 00H
и информация об аппаратной конфигурации в буфере
Если функция не выполнена
AH = код ошибки
Замечания. После возврата из прерывания буфер заполнен информацией об аппаратной конфигурации в следующем формате:

Смещение	Длина	Описание
00H	2	Размер исходных страниц дополнительной памяти (в параграфах)
02H	2	Число дополнительных наборов регистров
04H	2	Размер области сохранения контекста отображения (в байтах)
06H	2	Число наборов регистров, которые можно назначить каналам ПДП
08H	2	Тип операции ПДП (0 = ПДП можно использовать с дополнительными наборами регистров; 1 = доступен лишь один набор регистров ПДП)

Возвращаемый размер области сохранения контекста отображения совпадает с размером, возвращаемым при вызове Int 67H с функцией 4EH и подфункцией 03H.

Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H **[EMS 4.0]**
Функция 59H(89), подфункция 01H
Получить число исходных страниц

Возвращает полное число исходных страниц дополнительной памяти, присутствующих в системе, а также число еще невыделенных исходных страниц. Исходные страницы памяти могут иметь размер, отличающийся от 16 Кбайт.
При вызове. AH = 59H
AL = 01H
При возврате. Если функция выполнена успешно
AH = 00H
BX = число невыделенных исходных страниц
DX = полное число исходных страниц
Если функция не выполнена
AH = код ошибки
Замечание. Если диспетчер дополнительной памяти поддерживает только страницы

стандартного размера, значения, возвращаемые этой функцией, совпадают со значениями, возвращаемыми при вызове Int 67H с функцией 42H.

Int 67H **[EMS 4.0]**
Функция 5AH(90), подфункция 00H
Выделить дескриптор и стандартные страницы

Выделяет дескриптор EMM и связанные с этим дескриптором стандартные (16 Кбайт) страницы дополнительной памяти.
При вызове. AH = 5AH
AL = 00H
BX = число выделяемых стандартных страниц
При возврате. Если функция выполнена успешно
AH = 00H
DX = дескриптор EMM
Если функция не выполнена
AH = код ошибки
Замечание. В отличие от прерывания Int 67H с функцией 43H выделение этой функцией нулевого числа страниц не является ошибкой.

Int 67H **[EMS 4.0]**
Функция 5AH(90), подфункция 01H
Выделить дескриптор и исходные страницы

Выделяет исходный дескриптор EMM и связанные с этим дескриптором исходные страницы дополнительной памяти.
При вызове. AH = 5AH
AL = 01H
BX = число выделяемых исходных страниц
При возврате. Если функция выполнена успешно
AH = 00H
DX = дескриптор для исходных страниц EMM
Если функция не выполнена
AH = код ошибки
Замечания. Исходные страницы памяти могут иметь размер, отличный от 16 Кбайт.
Выделение нулевого числа страниц не является ошибкой.

Int 67H **[EMS 4.0]**
Функция 5BH(91), подфункция 00H
Получить дополнительные регистры отображения

Возвращает номер активного дополнительного набора регистров или, если отсутствуют активные дополнительные наборы, сохраняет состояние регистров отображения в буфере и возвращает его адрес.
При вызове. AH = 5BH
AL = 00H
При возврате. Если функция выполнена успешно и дополнительный набор регистров отображения активен
AH = 00H
BL = текущий активный набор регистров отображения

Если функция выполнена успешно и дополнительный набор регистров отображения неактивен
AH = 00H
BL = 00H
ES:DI = сегмент: относительный_адрес области сохранения дополнительного набора регистров отображения (если BL = 0)
Если функция не выполнена
AH = код ошибки

Замечания. Адрес области сохранения должен быть указан в предшествующем вызове Int 67H с функцией 5BH и подфункцией 01H; область сохранения должна быть инициализирована предшествующим вызовом Int 67H с функцией 4EH и подфункцией 00H. Если предшествующего вызова Int 67H с функцией 5BH и подфункцией 01H не было, возвращенный адрес равен 0 и регистры не сохраняются.

Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H **[EMS 4.0]**
Функция 5BH(91), подфункция 01H
Установить дополнительные регистры отображения

Выбирает дополнительный набор регистров отображения или (если дополнительные наборы не поддерживаются) восстанавливает из указанного буфера контекст отображения.

При вызове. AH = 5BH
AL = 01H
BL = номер дополнительного набора регистров или 00H
ES:DI = сегмент: относительный_адрес области восстановления регистров отображения (если BL = 0)

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечания. Адрес буфера, указываемый в этом вызове, возвращается последующими вызовами Int 67H с функцией 5BH и подфункцией 00H при BL = 00H.

Область сохранения должна быть инициализирована предшествующим вызовом Int 67H с функцией 4EH и подфункцией 00H.

Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H **[EMS 4.0]**
Функция 5BH(91), подфункция 02H
Получить размер области сохранения дополнительных регистров отображения

Возвращает объем памяти, необходимой для выполнения прерываний 67H с функцией 5BH и подфункциями 00H и 01H.

При вызове. AH = 5BH
AL = 02H

При возврате. Если функция выполнена успешно
AH = 00H
DX = размер буфера (в байтах)
Если функция не выполнена

AH = код ошибки

Замечание. Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H [EMS 4.0]
Функция 5BH(91), подфункция 03H
Выделить дополнительный набор регистров отображения

Выделяет дополнительный набор регистров отображения для использования при вызове Int 67H с функцией 5BH и подфункциями 00H и 01H. Содержимое текущих активных регистров отображения копируется в заново выделенные дополнительные регистры отображения с целью обеспечения начального контекста при их выборе.

При вызове. AH = 5BH
AL = 03H

При возврате. Если функция выполнена успешно
AH = 00H
BL = номер дополнительного набора регистров отображения или 0, если отсутствуют доступные дополнительные наборы
Если функция не выполнена
AH = код ошибки

Замечание. Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H [EMS 4.0]
Функция 5BH(91), подфункция 04H
Освободить дополнительный набор регистров отображения

Освобождает дополнительный набор регистров отображения, выделенный вызовом Int 67H с функцией 5BH и подфункцией 03H.

При вызове. AH = 5BH
AL = 04H
BL = номер дополнительного набора регистров

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечания. Текущий дополнительный набор регистров отображения не может быть освобожден.

Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H [EMS 4.0]
Функция 5BH(91), подфункция 05H
Выделить набор регистров ПДП

Выделяет набор регистров ПДП

При вызове. AH = 5BH
AL = 05H

При возврате. Если функция выполнена успешно

AH = 00H
BL = номер набора регистров ПДП (0 = нет доступных наборов)
Если функция не выполнена
AH = код ошибки

Замечание. Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H **[EMS 4.0]**
Функция 5BH(91), подфункция 06H
Разрешить ПДП через дополнительный набор регистров отображения

Связывает канал ПДП с дополнительным набором регистров отображения.
При вызове. AH = 5BH
AL = 06H
BL = дополнительный набор регистров отображения
DL = номер канала ПДП
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечания. Если канал ПДП не назначен определенному набору регистров, ПДП по этому каналу будет отображаться через текущий набор регистров.

Если в качестве дополнительного набора регистров отображения указан 0, никакие особые действия, связанные с прямым доступом по указанному каналу ПДП, не выполняются.

Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H **[EMS 4.0]**
Функция 5BH(91), подфункция 07H
Запретить ПДП через дополнительный набор регистров отображения

Запрещает прямой доступ по всем каналам ПДП, связанным с определенным дополнительным набором регистров отображения.
При вызове. AH = 5BH
AL = 07H
BL = номер дополнительного набора регистров
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечание. Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H **[EMS 4.0]**
Функция 5BH(91), подфункция 08H
Освободить набор регистров ПДП

Освобождает набор регистров ПДП, предварительно выделенный вызовом Int 67H с функцией 5BH и подфункцией 05H.

При вызове. AH = 5BH
AL = 08H
BL = номер набора регистров ПДП
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечание. Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

Int 67H **[EMS 4.0]**
Функция 5CH(92)
Подготовить диспетчер дополнительной памяти к теплой перезагрузке

Подготавливает аппаратуру дополнительной памяти к предстоящей теплой перегрузке. Эта функция влияет на контекст текущего отображения, используемый дополнительный набор регистров, а также на все остальные средства аппаратуры дополнительной памяти, которые обычно инициализируются в процессе загрузки системы.
При вызове. AH = 5CH
При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки
Замечание. Если прикладная программа отображает дополнительную память по адресам ниже 640 Кбайт, прикладной программе надлежит перехватывать все возможные условия, могущие привести к теплой перезагрузке, чтобы в первую очередь можно было вызвать эту функцию.

Int 67H **[EMS 4.0]**
Функция 5DH(93), подфункция 00H
Разрешить системные функции EMM

Разрешает вызов функций EMM, специфичных для операционной системы (Int 67H с функциями 59H, 5BH и 5DH), любыми программами или драйверами устройств. (Это состояние, действующее по умолчанию.)
При вызове. AH = 5DH
AL = 00H
BX:CX = ключ доступа (при не первом вызове этой функции)
При возврате. Если функция выполнена успешно
AH = 00H
BX:CX = ключ доступа (при первом вызове этой функции)
Если функция не выполнена
AH = код ошибки
Замечания. Ключ доступа возвращается в регистрах BX и CX при первом вызове Int 67H с функцией 5DH и подфункциями 00H или 01H. Ключ доступа требуется для всех последующих вызовов любой из этих функций.

Эта функция предназначена для использования исключительно операционными системами.

Int 67H **[EMS 4.0]**
Функция 5DH(93), подфункция 01H
Запретить системные функции EMM

Запрещает вызов функций EMM, специфичных для операционной системы (Int 67H с функциями 59H, 5BH и 5DH), прикладными программами и драйверами устройств, резервируя использование этих функций операционной системой.

При вызове. AH = 5DH
AL = 01H
BX:CX = ключ доступа (при не первом вызове этой функции)

При возврате. Если функция выполнена успешно
AH = 00H
BX:CX = ключ доступа (при первом вызове этой функции)
Если функция не выполнена
AH = код ошибки

Замечание. Ключ доступа возвращается в регистрах BX и CX при первом вызове Int 67H с функцией 5DH и подфункциями 00H или 01H. Ключ доступа требуется для всех последующих вызовов любой из этих функций.

Эта функция предназначена для использования исключительно операционными системами.

Int 67H **[EMS 4.0]**
Функция 5DH (93), подфункция 02H
Освободить ключ доступа

Освобождает ключ доступа, полученный предшествующим вызовом 67H с функцией 5DH и подфункциями 00H или 01H.

При вызове. AH = 5DH
AL = 02H
BX:CX = ключ доступа

При возврате. Если функция выполнена успешно
AH = 00H
Если функция не выполнена
AH = код ошибки

Замечания. По отношению к функциям дополнительной памяти, специфичным для операционной системы, EMM возвращается в состояние, существовавшее после инициализации системы. При следующем вызове Int 67H с функцией 5DH и подфункциями 00H или 01H возвращается новый ключ доступа.

Эта функция предназначена для использования исключительно операционными системами и может быть запрещена операционной системой в любой момент.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Адаптер монохромного дисплея/принтера (MDA) 72
Администратор представления OS/2 247
Адресация курсора 81
Атрибут FAR 31
– –, сравнение с NEAR 26
Атрибуты файлов 321
Атрибут NEAR 25
– –, сравнение с FAR 26
Атрибут STACK 30

Байт атрибута 82
– –, монохромный дисплей 82
– –, цветной дисплей 82
Библиотека динамической компоновки 258
Библиотекарь 38
Блок окружения 22
Блок параметров при вызове функций EXEC 172
Блок параметров BIOS 144
Блоки управления файлами 23
– – –, структура 144
Буфер регенерации 72

Ввод-вывод, отображенный на память 81
– – –, графический режим 85
– – –, текстовый режим 81
Ввод с клавиатуры 56
– –, без эха 60
– –, обработчики Ctrl-C и Ctrl-Break 62
– –, с помощью функций BIOS 61
– –, с эхом 60
– –, традиционный 59
– –, через дескрипторы 57
Вектор прерываний 18
Версии MS-DOS 11
– –, 1.0 11
– –, 1.25 11
– –, 2.00 11
– –, 2.11, 2.25 12
– –, 3.0 12
– –, 3.1 12
– –, 3.2, 3.3, 4.0 12
Видеоадаптер 72
Видеографическая матрица (VGA) 72
Видеосистема 72
–, адаптеры 72
Видеофункции ПЗУ BIOS 79
Внешние аппаратные прерывания 192
Внешние команды 15
Внутренние аппаратные прерывания 192
Внутренние команды 15
Вывод координат мыши 70

Главная загрузочная запись 153
Графическая плата Hercules 72

Двоичный режим 203
Дескрипторы сегментов памяти 249
Динамическое распределение памяти 158
Директива GROUP 37
Диски 142
–, карта логического тома 142
–, корневой каталог 148
–, область файлов 149
–, ошибки 119
–, программа начальной загрузки 16
–, разделы жесткого диска 153
–, сектор загрузчика 143
Диски, таблица размещения файлов 146
Диспетчер дополнительной памяти (EMM) 161
– – – –, коды ошибок 165
– – – –, проверка наличия 162
Драйвер ANSI.SYS 76
Драйвер CLOCK 221
Драйверы блочно-ориентированных устройств 202
Драйверы символьно-ориентированных устройств 202

Заголовок арены 155
Заголовок устройства 204
Заголовок файла .EXE-программы 29
Записи 103
–, использование дескрипторов 113
–, использование FCB 104

Инструментальные средства программирования 38
– – –, библиотекарь 51
– – –, оптимизированный компилятор C 42
– – –, пособия 55
– – –, типы файлов 39
– – –, утилита CREF 49
– – –, утилита EXE2BIN 48
– – –, утилита MAKE 53
– – –, LINK 44

– – –, MASM 40
Интерфейс API 248

Кадр страниц 161
Капсулирование подпрограмм 251
Каталог 133
– , иерархическая структура 134
– , корневой 134
– , перемещение файлов 139
– , поиск 135
– , формат записи 148
Ключи
– библиотекаря 52
– компилятора C 42
– компоновщика 46
– макроассемблера 41
– утилиты MAKE 54
Коды ошибок диспетчера дополнительной памяти 165
Коды ошибок драйвера устройства 207
Коды ошибок MS-DOS 117
Команда BACKUP 15
– CALL 38
– CHDIR 135
– CHKDSK 15
– COPY 15
– DEL 15
– DEVICE 14
– DIR 15
– ENDP 31
– ENDS 26
– MKDIR 135
– ORG 26
– PAGE 26
– POP 31
– PROC 26
– PUSH 31
– REN 15
– RESTORE 15
Команда RET 37
– RMDIR 135
– SEGMENT 26
– TITLE 26
– TREE 139
– VOL 139
Командный процессор 14
Компоновка с программами обработчиков Ctrl-C, Ctrl-Break 64
Компоновщик 34
– , ключи 47
Компьютер IBM PC 56
– IBM PC/AT 56
– IBM PS/2 56
Конец прерываний (EOI) 196
Корневой каталог 134
Корпорация IBM 9
Край экрана 80

Логический дисковод 133

Макроассемблер 38
– , интерактивный режим 40
– , ключи версии 5.1 41
– , модули 34
– , режим командной строки 40
– , сегменты 34
– , справочники 55
Метка тома 139
Микропроцессор Zilog Z-80 9
Многоцветная графическая матрица (MCGA) 72
Модели памяти 36
Модуль BIOS 13
Модуль SYSINIT 18
– – , расположение в памяти 19
– – , установка драйверов устройств 229

Область транзитных программ 15
Оболочка COMMAND.COM PLUS 14
Оболочка Windows 13
Обработка типичного запроса на ввод-вывод 220
Обработчики прерываний 16
Обработчики Ctrl-C, Ctrl-Break 62
Образ файла .COM 25
– – .EXE 28
Объектные модули 34
Оверлеи 189
Операционная система CP/M 9
Операционная система MS-DOS 9
– – –, загрузка 16
– – –, обработчики прерываний 196
– – –, происхождение 9
– – –, структура 14
Операционная система OS/2 13
– – –, оптимизация программ 257
– – –, перенос программ MS-DOS 248
– – –, совместимость 247
– – –, функциональные вызовы, совместимые с MS-DOS и BIOS 225
Операционная система P-system 9
Операционная система PC-DOS 9
Операционная система 86-DOS 9
Операционные системы UNIX/XENIX 57
Опрос мыши 70
Оптимизирующий компилятор C 42

Параллельные порты 87
Принтер 87
Программа начального загрузчика 16
Программа стратегии 204
Программируемый контроллер прерываний Intel 8259A 92
Программные модули 34
Программные прерывания 193
Программные процедуры 37
Программные сегменты 34
Программные файлы .COM 15
Программные файлы .EXE 28
– – –, заголовок 29
– – –, образ памяти 29

– – –, распределение памяти 156
– – –, содержимое загрузочного модуля 30
– – –, сравнение с .COM 24
– – –, формат загрузочного модуля 29
Программы командных кодов 208
Программы на языке ассемблера 33
Процедура, объявление начала/конца 26
Путь 134

Разделы жесткого диска 153
Расширенная память 168
– –, пересылка блоков данных 169
Расширенный блок управления файлом 106
Регистр дополнительного сегмента (ES) 30
Регистр программного сегмента (CS) 30
Регистр сегмента данных (DS) 30
Регистр сегмента стека (SS) 30
Регистр указателя стека 24
Режим ASCII 203
Резервные области диска 145
Резидентные драйверы устройств 14
Родительские программы 171

Сводка функций ПЗУ BIOS 391
Сегмент данных 35
Сегмент программы 35
Сегмент стека 35
Сегментный регистр 249
Сегменты памяти 249
Сектор загрузки 179
Семейство микропроцессоров Intel 80 × 80 9
Символ . 150
– | 232
– .. 150
– ; 52
– @ 52
– < 232
– > 232
– » 232
Системная таблица файлов 114
Слово атрибутов устройства 206
Совместимость и переносимость 243
Совместимость ПЗУ BIOS 243
Спецификация дополнительной памяти (EMS) 161
– – – –, использование памяти 164
– – – –, проверка наличия памяти 163
Стандартное последовательное устройство 21
Стандартное устройство вывода 21
Стандартное устройство печати 21
Стандарт RS-232 87
Стандарт RS-422 87
Степени совместимости 243
Строки ASCIIZ 22

Таблица векторов прерываний 195
Таблица размещения файлов (FAT) 134
Текстовые ASCII-файлы 49
Тип выравнивания 35
Тип класса 35
Тип объединения 35
Транзитные программы 16

Улучшенный графический адаптер (EGA) 72
Управление памятью 155
Управление экраном с помощью функций MS-DOS 76
Управляющие Esc-последовательности 77
– –, ассемблирование 221
– –, заголовок устройства 204
– –, коды ошибок 207
– –, компоновка 228
– –, написание 221
– –, обработка типичного запроса на ввод-вывод 220
– –, отладка 230
– –, программа прерывания 207
– –, программа стратегии 204
– –, программы командных кодов 208
– –, слово атрибутов устройства 206
– –, структура 204
– –, типы 202
– –, установка 228
– –, цепочки 229
– –, CLOCK 221
Устанавливаемые драйверы устройств 202
Устройства COM1, COM2, COM3 87
Устройства LPT1, LPT2, LPT3 87
Устройство стандартного ввода 21
Устройство стандартной ошибки 21
Устройство CLOCK$ 14
Устройство PRN 14
Утилита CREF 49
Утилита EXE2BIN 48
Утилита MAKE 53
– –, ключи 54

Файл, открытие 18
Файл COMMAND.COM 14
– –, загрузка 15
– –, замена 14
– –, использование функции EXEC 171
Файл CONFIG.SYS 14
Файл IBMBIO.COM 14
Файл IBMDOS.COM 14
Файл 10.SYS 16
– –, расположение в памяти 18
Файл MSDOS.SYS 14
– –, расположение в памяти 19
Файлы действия 53
Файлы
–, имена и расположение 134
–, логические диски 133
–, область на диске 149
–, перемещение 139
–, типы 39
–, ASM 39
–, C 39
–, CRF 39
–, DEF 39
–, H 39

– .INC 39
– .LIB 39
– .LST 39
– .MAP 39
– .OBJ 39
– .REF 39
Фильтры 231
Формат заголовка запроса 206
Функции клавиатуры ПЗУ BIOS 57
Функции MS-DOS 259
Функция EXEC 16
– –, блок окружения 172
– –, возврат из неё 174
– –, загрузка оверлеев 188
– –, получение памяти 171
– –, пример использования 175
– –, совместимость в приложениях MS-DOS 246
– –, соглашение вызова 174

Хвост команды 23

Цветной графический адаптер (CGA) 72

Чередование памяти 85

Элементы арены 155

Ядро DOS 14
Язык C 42

ОГЛАВЛЕНИЕ

Предисловие переводчика 5
Введение 7

Раздел 1. Программирование в MS-DOS . 9

Глава 1. Происхождение MS-DOS . . . 9
Глава 2. Система MS-DOS в действии 13
Глава 3. Структура прикладных программ в среде MS-DOS 21
Глава 4. Инструментальные средства программирования в системе MS-DOS . 38
Глава 5. Ввод с помощью клавиатуры и манипулятора мышь . . 56
Глава 6. Видеосистема 72
Глава 7. Принтер и последовательный порт . 87
Глава 8. Управление файлами . . 103
Глава 9. Тома и каталоги 133
Глава 10. Внутренняя структура диска . 142
Глава 11. Управление памятью . . 155
Глава 12. Функция EXEC 171
Глава 13. Обработчики прерываний 190
Глава 14. Устанавливаемые драйверы устройств . . . 202
Глава 15. Фильтры 231
Глава 16. Совместимость и переносимость . . 243

Раздел 2. Справочник по функциям MS-DOS . 259

Раздел 3. Справочник по функциям ПЗУ BIOS фирмы IBM и манипулятора мышь 391

Раздел 4. Справочник по функциям спецификации дополнительной памяти LOTUS/INTEL/MICROSOFT EMS 481

Предметный указатель . 506

УВАЖАЕМЫЙ ЧИТАТЕЛЬ!

Ваши замечания о содержании книги, ее оформлении, качестве перевода и другие просим присылать по адресу: 129820, Москва, И-110, 1-й Рижский пер., д. 2, изд-во «Мир».

Научное издание

Рей Данкан

ПРОФЕССИОНАЛЬНАЯ РАБОТА В MS-DOS

Заведующий редакцией Э. Н. Бадиков
Научный редактор С. В. Богдасаров
Мл. научный редактор М. Ю. Григоренко
Художник В. И. Потапов
Художественные редакторы Н. М. Иванов, О. Н. Адаскина
Технический редактор Т. А. Максимова
Корректор С. С. Суставова

ИБ № 7703

Сдано в набор 19.04.91. Подписано к печати 22.10.92. Формат 70 × $100^{1}/_{16}$. Бумага офсетная № 2. Печать офсетная. Гарнитура таймс. Объем 16,00 бум. л. Усл. печ. л. 41,60. Усл. кр.-отт. 43,53. Уч.-изд. л. 43,51. Изд. № 6/7854. Тираж 30 000 экз. Зак. 436. С012

Издательство «Мир», Министерства печати и информации Российской Федерации, 129820, ГСП, Москва, И-110, 1-й Рижский пер., 2.

Можайский полиграфкомбинат Министерства печати и информации Российской Федерации, г. Можайск, ул. Мира, 93.